Welcome to the Windows API Guide! Here, you will discover a wealth of information to help you implement calls to the Windows API directly from your Visual Basic programs. This site is sort of a work in progress; it falls short of encompassing the entire Windows API. Nevertheless, new information about functions and the like is added every two or three weeks. Check back often to see what's new!

The New Guru Is Here!

Hello fellow programmers! I'm Chris Pietschmann, and I will be updating this site from now on. There isn't anything new yet, but I'm working on a new format for the site. All the old stuff will remain in the old look, and Only new stuff will have the new look. But I will still update the old stuff.

Reference

The Reference section contains documentation on hundreds of Windows API functions. Besides the functions, information on the related structures and other items also appears for reference. Each function page includes a well-commented example illustrating common usage of the function.

Articles

The Articles section contains articles about API-related topics. These multi-page articles go in-depth about a specific issue in the API and offer a more well-rounded approach to learning about API functions without much technical information. Of course, the reference information for each function is only a click away.

Still can't find what you're looking for anywhere on the site? Then try browsing the VB-World.net web site. Along with their sizable collection of Visual Basic programming information, they also have an API programming section filled with helpful articles, some of which cover ideas not currently discussed on this web site.
Still unable to find what you're looking for? You might want to e-mail me about what you're looking for. If I can't give you the information, I might be able to point you in the right direction anyway.

Awards the Windows API Guide has won:

![Cool Site](http://www.vbapi.com/images/CoolSite.gif)

Last Modified: March 7, 2002
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/index.html
Windows API Reference

Last Update: March 2, 2001

Important Message
I, Paul Kuliniewicz, am no longer able to continue maintaining this web site. There will be no more updates to this site unless some other party takes over operation. Please read my farewell message for all the details and other information for loyal visitors.

Welcome to the Reference section of the Windows API Guide. Here, you will find documentation on hundreds of API functions and the structures that go along with them. Each API function page includes an example in Visual Basic 5.0 which demonstrates proper usage. The information in the reference section is organized according to what it describes. Please click on a link below.

Interested in what has been added or edited since the last update of the Windows API Guide? Take a look at the Revision History of the Reference section!

January 21, 2001: A smaller update than usual. The examples on all the Winsock functions have been fixed. The old code could hang if the remote server would stop responding. That bug has been corrected via the ioctlsocket function. Some list box messages have also been added, a function or two that work with screen resolutions, and a number of various error fixes. Sadly, problems with the server prevented me from getting the new search engine ready for this update. It should be up and running next time. Speaking of which, there will be some revisions in the overall layout of the site, which will hopefully aid navigation. Be warned: any bookmarks that point to anything that isn't http://www.vbapi.com/ will probably break, so be prepared.

- **Function Information:** (357 functions listed, 4 newly added)
  - Alphabetical Listing
  - Categorical Listing
- **Structure Information:** (83 structures listed)
  - Alphabetical Listing
- **Message Information:** (66 messages listed, 6 newly added)
The following list identifies which pages have been added or significantly changed in each update of the Windows API Guide's Reference section since February 13, 2000. Added or edited pages are listed according to date and general description (i.e., function pages are grouped together, etc.). Any page that was added in the update is flagged with NEW. Any page that was edited significantly (such as a correction or a layout change) appears in normal text.

January 21, 2001 Update

January 21, 2001: A smaller update than usual. The examples on all the Winsock functions have been fixed. The old code could hang if the remote server would stop responding. That bug has been corrected via the ioctlsocket function. Some list box messages have also been added, a function or two that work with screen resolutions, and a number of various error fixes. Sadly, problems with the server prevented me from getting the new search engine ready for this update. It should be up and running next time. Speaking of which, there will be some revisions in the overall layout of the site, which will hopefully aid navigation. Be warned: any bookmarks that point to anything that isn't http://www.vbapi.com/ will probably break, so be prepared.

- Functions:
  - ChangeDisplaySettings NEW
  - ClosePrinter
  - closesocket
  - connect
  - EnumDisplaySettings NEW
  - GetProfileInt
  - htons
  - ioctlsocket NEW
  - OpenPrinter
  - PrinterProperties NEW
  - recv
Windows API Guide: Reference: Revision History

- RegQueryValueEx
- send
- SetFilePointer
- SHGetSpecialFolderLocation
- socket

- **Structures:**
  - DEVMODE

- **Messages:**
  - LB_GETCURSEL NEW
  - LB_GETSEL NEW
  - LB_GETSELCOUNT NEW
  - LB_GETSELITEMS NEW
  - LB_SETCURSEL NEW
  - LB_SETSEL NEW

- **Other:**
  - Extended Window Styles

---

**December 17, 2000 Update**

**December 17, 2000:** Some more Winsock functions have been added, along with a few list box messages. I've also corrected a few minor mistakes on some pages, but I haven't yet been able to fix all the mistakes I've been notified of yet. Hopefully those will be done in time for the next update. Finally, I'm still working on the search engine for the site. Most of it is done, but it still needs some more testing and security checking before I add it to the site.

- **Functions:**
  - closesocket NEW
  - connect NEW
  - DestroyWindow
  - FindWindow
  - FindWindowEx
  - GetFocus NEW
  - GetSaveFileName
  - htonl
  - htons NEW
  - KillTimer
  - LockWorkStation NEW
  - recv NEW
  - RemoveMenu NEW
October 29, 2000 Update

October 29, 2000: I've finally added a few Winsock API functions, at popular request. Also making an appearance are the CreateWindowEx function and information about the IP Address control. Finally, I've moved window class information into its own small section.

Functions:
- CreateWindowEx NEW
- DestroyWindow NEW
- GetClassLong
- gethostbyaddr NEW
- gethostbyname NEW
- gethostname NEW
- GetWindowLong
- htonl NEW
- inet_addr NEW
- inet_ntoa NEW
- InitCommonControlsEx NEW
Windows API Guide: Reference: Revision History

- **SendInput**
- **SetWindowLong**
- **WSACleanup** NEW
- **WSAGetLastError** NEW
- **WSAStartup** NEW

- **Structures:**
  - **ENUMLOGFONT**
  - **HOSTENT** NEW
  - **INITCOMMONCONTROLSEX_TYPE** NEW
  - **WSADATA** NEW

- **Messages:**
  - **BM_CLICK**
  - **IPM_CLEARADDRESS** NEW
  - **IPM_GETADDRESS** NEW
  - **IPM_ISBLANK** NEW
  - **IPM_SETADDRESS** NEW
  - **IPM_SETFOCUS** NEW
  - **IPM_SETRANGE** NEW

- **Macros:**
  - **FIRST_IPADDRESS** NEW
  - **FOURTH_IPADDRESS** NEW
  - **HIBYTE** NEW
  - **LOBYTE** NEW
  - **MAKEIPRANGE** NEW
  - **MAKELANGID**
  - **MAKEWORD** NEW
  - **SECOND_IPADDRESS** NEW
  - **THIRD_IPADDRESS** NEW

- **Window Classes:**
  - **Base Window Styles** NEW
  - **Button Control Window Class** NEW
  - **Combo Box Control Window Class** NEW
  - **Edit Control Window Class** NEW
  - **Extended Window Styles** NEW
  - **IP Address Control Window Class** NEW
  - **List Box Control Window Class** NEW
  - **Scroll Bar Control Window Class** NEW
  - **Static Control Window Class** NEW

- **Other:**
September 24, 2000 Update

September 24, 2000: Starting today, I'm going to write a short message like this in each update. For all of you who've asked me about how to get a list of the currently running processes on the system, your prayers have been answered. At least, the prayers about getting a list of processes on the system.... Anyway, you'll want to look at CreateToolhelp32Snapshot for that. I also added, among other things, SetWindowRgn, needed to make nonrectangular-looking windows. For the complete list of new goodies, go to the history page.

- Functions:
  - CreateToolhelp32Snapshot NEW
  - GetDiskFreeSpaceEx
  - GetOpenFileName
  - GetSaveFileName
  - GetSysColor NEW
  - GetWindowRgn NEW
  - Process32First NEW
  - Process32Next NEW
  - RegDeleteKey
  - RegEnumValue
  - SetSysColors NEW
  - SetWindowRgn NEW

- Structures:
  - OPENFILENAME
  - PROCESSENTRY32 NEW

- Messages:
  - CB_ADDSTRING NEW
  - CB_DELETESTRING NEW
  - CB_INSERTSTRING NEW
  - CB_RESETCONTENT NEW

- Callback Functions:
  - OFNHookProc NEW
  - OFNHookProcOldStyle NEW

August 26, 2000 Update
Windows API Guide: Reference: Revision History

- Functions:
  - GetCapture
  - GetEnvironmentVariable NEW
  - mouse_event
  - QueryPerformanceCounter NEW
  - QueryPerformanceFrequency NEW
  - SendInput
  - SetEnvironmentVariable NEW
  - ShellExecute
  - ShellExecuteEx NEW
  - WaitForSingleObject NEW

- Structures:
  - ITEMIDLIST
  - JOB_INFO_1
  - LARGE_INTEGER NEW
  - LOGFONT
  - SHELLEXECUTEINFO NEW

- Messages:
  - EM_CANUNDO NEW
  - EM_GETFIRSTVISIBLELINE NEW
  - EM_GETLINE NEW
  - EM_LINEINDEX NEW
  - EM_LINELENGTH NEW
  - EM_UNDO NEW

- MCI Commands:
  - open

July 30, 2000 Update

- Functions:
  - EqualRgn
  - GetAsyncKeyState
  - GetFileVersionInfo NEW
  - GetFileVersionInfoSize NEW
  - GetKeyState
  - RegCloseKey
  - RegCreateKeyEx
  - RegisterClassEx
Windows API Guide: Reference: Revision History

● RegQueryValueEx
● SetTimer
● VerQueryValue NEW

● Structures:
  ○ VS_FIXEDFILEINFO NEW

● Messages:
  ○ BM_CLICK NEW
  ○ BM_GETCHECK NEW
  ○ BM_GETSTATE NEW
  ○ BM_SETCHECK NEW
  ○ BM_SETSTATE NEW

● Macros:
  ○ GET_X_LPARAM
  ○ GET_Y_LPARAM
  ○ HIWORD
  ○ LOWORD

July 4, 2000 Update

● Functions:
  ○ ExitWindowsDialog NEW
  ○ GetShortPathName
  ○ mciGetErrorString NEW
  ○ mciSendString NEW
  ○ RestartDialog NEW

● Messages:
  ○ MM_MCINOTIFY NEW

● MCI Command Strings:
  ○ close NEW
  ○ open NEW
  ○ pause NEW
  ○ play NEW
  ○ stop NEW

June 4, 2000 Update

● Functions:
  ○ CreatePopupMenu NEW
Windows API Guide: Reference: Revision History

- **DestroyMenu** NEW
- **GetMenu** NEW
- **GetMenuItemCount** NEW
- **GetMenuItemInfo** NEW
- **GetSystemMenu** NEW
- **GetVolumeInformation**
- **InsertMenuItem** NEW
- **PickIconDlg** NEW
- **SetMenuItemInfo** NEW
- **TrackPopupMenu** NEW
- **TrackPopupMenuEx** NEW

**Structures:**

- **MENUITEMINFO** NEW
- **RECT**
- **TPMPARAMS** NEW

**Messages:**

- **WM_COMMAND** NEW
- **WM_INITMENU** NEW
- **WM_SYSCOMMAND** NEW

**Macros:**

- **HIWORD** NEW
- **LOWORD** NEW

May 21, 2000 Update

**Functions:**

- **GetCapture** NEW
- **KillTimer** NEW
- **RegEnumValue**
- **ReleaseCapture** NEW
- **SetCapture** NEW
- **SetTimer** NEW
- **SHFileOperation**

**Messages:**

- **EM_GETPASSWORDCHAR** NEW
- **EM_GETSEL** NEW
- **EM_REPLACESEL** NEW
- **EM_SETPASSWORDCHAR** NEW
April 16, 2000 Update

- Functions:
  - GetCurrencyFormat NEW
  - GetNumberFormat NEW
  - GetVolumeInformation NEW
  - RoundRect
  - SetVolumeLabel NEW
  - SHFileOperation NEW
  - SHFreeNameMappings NEW

- Structures:
  - CURRENCYFMT NEW
  - MEMORYSTATUSEX
  - NUMBERFMT NEW
  - SHFILEOPSTRUCT NEW
  - SHNAMEMAPPING NEW

- Messages:
  - CB_GETCOUNT NEW
  - CB_GETCURSEL NEW
  - CB_GETDROPPEDSTATE NEW
  - CB_GETLBTEXT NEW
  - CB_GETLBTEXTLEN NEW
  - CB_SETCURSEL NEW
  - CB_SHOWDROPDOWN NEW

- Macros:
  - MAKELCID NEW

March 19, 2000 Update

- Functions:
  - CreateDirectory
Windows API Guide: Reference: Revision History

- **CreateDirectoryEx**
- **GetAsyncKeyState**
- **GetDiskFreeSpaceEx**
- **GetTickCount** **NEW**
- **GlobalMemoryStatus** **NEW**
- **GlobalMemoryStatusEx** **NEW**
- **RegEnumKeyEx**
- **SendInput**
- **Shell_NotifyIcon** **NEW**

**Structures:**
- **MEMORYSTATUS** **NEW**
- **MEMORYSTATUSEX** **NEW**
- **NOTIFYICONDATA** **NEW**
- **POINT_TYPE**
- **ULARGE_INTEGER**

**Messages:**
- **WM_LBUTTONDBLCLK** **NEW**
- **WM_LBUTTONDOWN** **NEW**
- **WM_LBUTTONUP** **NEW**
- **WM_MBUTTONDBLCLK** **NEW**
- **WM_MBUTTONDOWN** **NEW**
- **WM_MBUTTONUP** **NEW**
- **WM_MOUSEMOVE** **NEW**
- **WM_RBUTTONDBLCLK** **NEW**
- **WM_RBUTTONDOWN** **NEW**
- **WM_RBUTTONUP** **NEW**

**Macros:**
- **GET_X_LPARAM** **NEW**
- **GET_Y_LPARAM** **NEW**
- **MAKELANGID**
- **MAKEPOINTS** **NEW**

**Other:**
- **Error Codes**

**February 13, 2000 Update**

**Functions:**
- **GetWindowText**
● GetWindowTextLength
● MessageBox NEW
● MessageBoxEx NEW
● MessageBoxIndirect NEW
● SetWindowText

● Structures:
  ○ HELPINFO NEW
  ○ MSGBOXPARAMS NEW

● Messages:
  ○ WM_GETTEXT NEW
  ○ WM_GETTEXTLENGTH NEW
  ○ WM_HELP NEW
  ○ WM_SETTEXT NEW

● Callback Functions:
  ○ MsgBoxCallback NEW

● Macros:
  ○ MAKELANGID NEW
Below is an alphabetical list of the API functions currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.
- Chord
- ClipCursor
- CloseHandle
- ClosePrinter
- closesocket
- CombineRgn
- CommDlgExtendedError
- CompareFileTime
- CompareString
- connect
- CopyFile
- CopyMemory
- CopyRect
- CoTaskMemFree
- CreateCursor
- CreateDC
- CreateDirectory
- CreateDirectoryEx
- CreateEllipticRgn
- CreateEllipticRgnIndirect
- CreateFile
- CreateFont
- CreateFontIndirect
- CreateHatchBrush
- CreatePen
- CreatePenIndirect
- CreatePolygonRgn
- CreatePolyPolygonRgn
- CreatePopupMenu
- CreateRectRgn
- CreateRectRgnIndirect
- CreateRoundRectRgn
- CreateSolidBrush
- CreateToolhelp32Snapshot
- CreateWindowEx

- DefWindowProc
- DeleteDC
- DeleteFile
Windows API Guide: Reference: Functions

- **DeleteObject**
- **DestroyCursor**
- **DestroyIcon**
- **DestroyMenu**
- **DestroyWindow**
- **DrawIcon**
- **DrawIconEx**

- **Ellipse**
- **EnableWindow**
- **EndDoc**
- **EndPage**
- **EnumChildWindows**
- **EnumDisplaySettings** **NEW**
- **EnumFontFamilies**
- **EnumFontFamiliesEx**
- **EnumJobs**
- **EnumPrinters**
- **EnumPropsEx**
- **EnumThreadWindows**
- **EnumWindows**
- **EqualRect**
- **EqualRgn**
- **ExitWindowsDialog**
- **ExitWindowsEx**
- **ExtFloodFill**
- **ExtractIcon**
- **ExtractIconEx**

- **FileTimeToLocalFileTime**
- **FileTimeToSystemTime**
- **FillMemory**
- **FillRect**
- **FillRgn**
- **FindClose**
- **FindFirstFile**
- **FindNextFile**
- **FindWindow**
- **FindWindowEx**
FlashWindow
FrameRect
FrameRgn

G

GetActiveWindow
GetArcDirection
GetAsyncKeyState
GetBrushOrgEx
GetCapture
GetClassInfo
GetClassInfoEx
GetClassLong
GetClassName
GetClipCursor
GetComputerName
GetCurrencyFormat
GetCursor
GetCursorPos
GetDateFormat
GetDC
GetDesktopWindow
GetDiskFreeSpace
GetDiskFreeSpaceEx
GetDoubleClickTime
GetDriveType
GetEnvironmentVariable
GetFileAttributes
GetFileInformationByHandle
GetFileSize
GetFileTime
GetFileVersionInfo
GetFileVersionInfoSize
GetFocus
GetForegroundWindow
GetFullPathName
gethostbyaddr
gethostbyname
gethostname
GetKeyboardState
GetKeyState
GetLastError
GetLocalTime
GetLogicalDrives
GetLogicalDriveStrings
GetMenu
GetMenuItemCount
GetMenuItemInfo
GetNumberFormat
GetOpenFileName
GetParent
GetPixel
GetPolyFillMode
GetPrivateProfileInt
GetPrivateProfileString
GetProfileInt
GetProfileString
GetProp
GetRgnBox
GetSaveFileName
GetShortPathName
GetStockObject
GetSysColor
GetSystemDirectory
GetSystemMenu
GetSystemMetrics
GetSystemTime
GetSystemTimeAsFileTime
GetTempFileName
GetTempPath
GetTextAlign
GetThreadLocale
GetTickCount
GetTimeFormat
GetTimeZoneInformation
GetTopWindow
GetUserName
GetVersionEx
- GetVolumeInformation
- GetWindow
- GetWindowLong
- GetWindowRect
- GetWindowsDirectory
- GetWindowRgn
- GetWindowText
- GetWindowTextLength
- GetWindowThreadProcessId
- GlobalAlloc
- GlobalFree
- GlobalLock
- GlobalMemoryStatus
- GlobalMemoryStatusEx
- GlobalUnlock
- htonl
- htons
- inet_addr
- inet_ntoa
- InflateRect
- InitCommonControlsEx
- InsertMenuItem
- IntersectRect
- InvertRect
- InvertRgn
- ioctlsocket NEW
- IsChild
- IsIconic
- IsRectEmpty
- IsWindow
- IsWindowEnabled
- IsZoomed
- joyGetDevCaps
- joyGetNumDevs
- joyGetPos
- keybd_event
- KillTimer

- L
  - LineTo
  - LoadCursor
  - LoadCursorFromFile
  - LocalFileTimeToFileTime
  - LockWorkStation
  - lstrcmp
  - lstrcmpi
  - lstrcpy
  - lstrcpyn
  - lstrlen

- M
  - mciGetErrorString
  - mciSendString
  - MessageBeep
  - MessageBox
  - MessageBoxEx
  - MessageBoxIndirect
  - mouse_event
  - MoveFile
  - MoveMemory
  - MoveToEx
  - MoveWindow
  - MulDiv

- N
- O

- P
  - PickIconDlg
  - Pie
  - PlaySound
  - PolyBezier
  - PolyBezierTo
  - Polygon
  - Polyline
- PolylineTo
- PolyPolygon
- PolyPolyline
- PrintDlg
- PrinterProperties \textbf{NEW}
- Process32First
- Process32Next
- PtInRect
- PtInRegion

- Q
- QueryPerformanceCounter
- QueryPerformanceFrequency

- R
- ReadFile
- Rectangle
- RectInRegion
- recv
- RegCloseKey
- RegCreateKeyEx
- RegDeleteKey
- RegDeleteValue
- RegEnumKeyEx
- RegEnumValue
- RegisterClass
- RegisterClassEx
- RegOpenKeyEx
- RegQueryValueEx
- RegSetValueEx
- ReleaseCapture
- ReleaseDC
- RemoveDirectory
- RemoveMenu
- RemoveProp
- RestartDialog
- RoundRect

- S
- SelectObject
- send
- SendInput
- SendMessage
- SetActiveWindow
- SetArcDirection
- SetBrushOrgEx
- SetCapture
- SetClassLong
- SetCursor
- SetCursorPos
- SetDoubleClickTime
- SetEnvironmentVariable
- SetFileAttributes
- SetFilePointer
- SetFileName
- SetFocus
- SetForegroundWindow
- SetKeyboardState
- SetLastError
- SetLastErrorEx
- SetMenuItemInfo
- SetParent
- SetPixel
- SetPixelV
- SetPolyFillMode
- SetProp
- SetRect
- SetRectEmpty
- SetSysColors
- SetSystemCursor
- SetSystemTime
- SetTextAlign
- SetThreadLocale
- SetTimer
- SetVolumeLabel
- SetWindowLong
- SetWindowPos
- SetWindowRgn
- SetWindowText
- SHAddToRecentDocs
- SHBrowseForFolder
- Shell_NotifyIcon
- ShellExecute
- ShellExecuteEx
- SHEmptyRecycleBin
- SHFileOperation
- SHFreeNameMappings
- SHGetFileInfo
- SHGetFolderLocation
- SHGetFolderPath
- SHGetPathFromIDList
- SHGetSpecialFolderLocation
- SHGetSpecialFolderPath
- ShowCursor
- ShowWindow
- SHQueryRecycleBin
- SHUpdateRecycleBinIcon
- Sleep
- sndPlaySound
- socket
- StartDoc
- StartPage
- StretchBlt
- SubtractRect
- SwapMouseButton
- SystemParametersInfo
- SystemTimeToFileTime
- TextOut
- TrackPopupMenu
- TrackPopupMenuEx
- UnionRect
- UnregisterClass
- VerQueryValue
- WaitForSingleObject
- waveOutGetDevCaps
waveOutGetNumDevs
waveOutGetVolume
waveOutSetVolume
WindowFromPoint
WinHelp
WriteFile
WritePrivateProfileString
WriteProfileString
WSACleanup
WSAGetLastError
WSAStartup

X
Y
Z

ZeroMemory
AngleArc Function

Declare Function AngleArc Lib "gdi32.dll" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal dwRadius As Long, ByVal eStartAngle As Single, ByVal eSweepAngle As Single) As Long

Platforms

- **Windows 95**: Not Supported
- **Windows 98**: Not Supported
- **Windows NT**: Requires Windows NT 3.1 or greater
- **Windows 2000**: Supported
- **Windows CE**: Not Supported

Description & Usage

AngleArc draws a circular arc on a device using the device's current pen. The circle which the arc lies on is determined by its center and radius. The start and end points of the arc are determined by angle measures in degrees, measured counterclockwise from the line parallel to the positive x-axis (i.e., from due right). The arc itself is drawn either clockwise or counterclockwise to connect the points, depending on the device's settings. AngleArc also draws a line connecting the device's current point to the beginning of the arc.

Return Value

If an error occurs, the function returns 0 (call GetLastError to get the error code). If the function succeeds, the function returns a non-zero value.

Visual Basic-Specific Issues

None

Parameters

- **hdc**: A handle to a device context of the device to draw the arc on.
- **x**: The x coordinate of the center of the circle.
- **y**: The y coordinate of the center of the circle.
Windows API Guide: AngleArc Function

dwRadius
The radius of the circle.
eStartAngle
The angle (in degrees) identifying the starting point of the arc.
eSweepAngle
The angle (in degrees) identifying the ending point of the arc.

Example

' This code is licensed according to the terms and conditions listed here.

' Draw an arc formed by the upper half of a circle (from 0 to 180 ' degrees counterclockwise). The circle is centered at (100, 150) and has a radius ' of 50. The arc is drawn using the solid black stock pen.
Dim hpen As Long ' handle to the black stock pen
Dim holdpen As Long ' handle to Form1's previously selected pen
Dim retval As Long ' return value

' Get the solid black stock pen and select it for use in Form1.
hpen = GetStockObject(BLACK_PEN) ' get the pen's handle
holdpen = SelectObject(Form1.hDC, hpen) ' select the pen

' Make sure arcs are drawn going counterclockwise
retval = SetArcDirection(Form1.hDC, AD_COUNTERCLOCKWISE)
' Draw the arc
retval = AngleArc(Form1.hDC, 100, 150, 50, 0, 180)

' Select Form1's previous pen to restore the "defaults".
retval = SelectObject(Form1.hDC, holdpen) ' select the old pen

See Also

Arc, ArcTo, Ellipse, GetArcDirection, SetArcDirection

Category

Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: July 27, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
Arc Function

Declare Function Arc Lib "gdi32.dll" (ByVal hdc As Long, ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long, ByVal nXStartArc As Long, ByVal nYStartArc As Long, ByVal nXEndArc As Long, ByVal nYEndArc As Long) As Long

Platforms

- **Windows 95**: Supported
- **Windows 98**: Supported
- **Windows NT**: Requires Windows NT 3.1 or later
- **Windows 2000**: Supported
- **Windows CE**: Not Supported

Description & Usage

Arc draws an elliptical arc on a device using the device's current pen. The ellipse which the arc lies on is inscribed within the bounding rectangle coordinates passed to the function. The start and end points are determined by two radials. The radials begin at the center of the ellipse and extend through the given radial point (either the start or end one); where they intersect the ellipse is where the start and end points of the arc are. **Windows 95/98**: The arc is always drawn counterclockwise from the start point to the end point. **Windows NT/2000**: The direction the arc is drawn in depends on the device's current setting.

Return Value

If an error occurred while attempting to draw the arc, the function returns 0 (**Win NT/2000**: call GetLastError to get the error code). If the function completed successfully, it returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hdc**
  A handle to a device context of the device to draw the arc on.
- **nLeftRect**
  The x coordinate of the upper-left point of the ellipse's bounding rectangle.
- **nTopRect**
  The y coordinate of the upper-left point of the ellipse's bounding rectangle.
nRightRect
The x coordinate of the lower-right point of the ellipse's bounding rectangle.

nBottomRect
The y coordinate of the lower-right point of the ellipse's bounding rectangle.

nXStartArc
The x coordinate of the radial point that determines the arc's starting point.

nYStartArc
The y coordinate of the radial point that determines the arc's starting point.

nXEndArc
The x coordinate of the radial point that determines the arc's ending point.

nYEndArc
The y coordinate of the radial point that determines the arc's ending point.

Example

' This code is licensed according to the terms and conditions listed here.

' Draw the arc that forms the top half of an ellipse. The ellipse
' is centered at (100, 100), has a width of 200, and has a height of 100. The arc is drawn
' on window Form1 using the black solid stock pen.
Dim hpen As Long  ' handle to the black solid pen
Dim holdpen As Long  ' handle to window Form1's previously selected pen
Dim retval As Long  ' return value

' Get a handle to the solid black pen and set it as Form1's drawing pen.
  hpen = GetStockObject(BLACK_PEN)  ' get a handle to the pen
  holdpen = SelectObject(Form1.hDC, hpen)  ' set it as Form1's current pen

' The ellipse is determined by the bounding rectangle (0,50)-(200,150).
' The radial to (200, 100) is due right; the radial to (0, 100) is due left.
retval = Arc(Form1.hDC, 0, 50, 200, 150, 200, 100, 0, 100)

' Restore Form1's previous pen selection.
retval = SelectObject(Form1.hDC, holdpen)  ' set the old pen back

See Also

AngleArc, ArcTo, Ellipse, GetArcDirection, SetArcDirection

Category

Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.
ArcTo Function

Declare Function ArcTo Lib "gdi32.dll" (ByVal hdc As Long, ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long, ByVal nXRadial1 As Long, ByVal nYRadial1 As Long, ByVal nXRadial2 As Long, ByVal nYRadial2 As Long) As Long

Platforms

- **Windows 95**: Not Supported
- **Windows 98**: Not Supported
- **Windows NT**: Requires Windows NT 3.1 or later
- **Windows 2000**: Supported
- **Windows CE**: Not Supported

Description & Usage

ArcTo draws an elliptical arc on a [device](#) using the device's current [pen](#). After drawing the arc, the device’s current point is set to the end point of the arc. The ellipse which the arc lies on is inscribed within the bounding rectangle coordinates passed to the function. The start and end points are determined by two radials. The radials begin at the center of the ellipse and extend through the given radial point (either the start or end one); where they intersect the ellipse is where the start and end points of the arc are. The direction the arc is drawn in depends on the device's current setting.

Return Value

If an error occurred while attempting to draw the arc, the function returns 0 (call [GetLastError](#) to get the error code). If the function completed successfully, it returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hdc**
  A handle to a device context of the device to draw the arc on.
- **nLeftRect**
  The x coordinate of the upper-left point of the ellipse's bounding rectangle.
- **nTopRect**
  The y coordinate of the upper-left point of the ellipse's bounding rectangle.
nRightRect
The x coordinate of the lower-right point of the ellipse's bounding rectangle.

nBottomRect
The y coordinate of the lower-right point of the ellipse's bounding rectangle.

nXRadial1
The x coordinate of the radial point that determines the arc's starting point.

nYRadial1
The y coordinate of the radial point that determines the arc's starting point.

nXRadial2
The x coordinate of the radial point that determines the arc's ending point.

nYRadial2
The y coordinate of the radial point that determines the arc's ending point.

Example

' This code is licensed according to the terms and conditions listed here.

' Draw the arc that forms the top half of an ellipse. The ellipse
' is centered at (100, 100), has a width of 200, and has a height of 100. The arc is
drawn
' on window Form1 using the black solid stock pen.
Dim hpen As Long  ' handle to the black solid pen
Dim holdpen As Long  ' handle to window Form1's previously selected pen
Dim retval As Long  ' return value

' Get a handle to the solid black pen and set it as Form1's drawing pen.
hpen = GetStockObject(BLACK_PEN)  ' get a handle to the pen
holdpen = SelectObject(Form1.hDC, hpen)  ' set it as Form1's current pen

' The ellipse is determined by the bounding rectangle (0,50)-(200,150).
' The radial to (200, 100) is due right; the radial to (0, 100) is due left.
retval = ArcTo(Form1.hDC, 0, 50, 200, 150, 200, 100, 0, 100)

' Restore Form1's previous pen selection.
retval = SelectObject(Form1.hDC, holdpen)  ' set the old pen back

See Also

AngleArc, Arc, Ellipse, GetArcDirection, SetArcDirection

Category

Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.
auxGetDevCaps Function

Declare Function auxGetDevCaps Lib "winmm.dll" Alias "auxGetDevCapsA"
(ByVal uDeviceID As Long, lpCaps As AUXCAPS, ByVal cbCaps As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The **auxGetDevCaps** function retrieves information about the capabilities of an auxiliary audio device. This function can also retrieve capabilities information about the auxiliary audio mapper, if one exists.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

**uDeviceID**

The device ID of the auxiliary audio device. Valid IDs range from 0 to the total number of auxiliary audio devices minus one. This could also be the following flag:

AUX_MAPPER

Retrieve information about the capabilities of the auxiliary audio mapper.

**lpCaps**
Receives information about the device's capabilities.

\textit{cbCaps}  
The size in bytes of the structure passed as \textit{lpCaps}.

\section*{Constant Definitions}

\texttt{Const AUX\_MAPPER = -1}

\section*{Example}

\verbatimverbatim{vbapi/ref/a/auxgetdevcaps.html}

\section*{Category}

\texttt{http://216.26.168.92/vbapi/ref/a/auxgetdevcaps.html}
auxGetNumDevs Function

Declare Function auxGetNumDevs Lib "winmm.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

auxGetNumDevs determines how many auxiliary audio devices are installed on the computer. This could be less than the number of devices actually present on the computer, since it is possible for a device to be installed yet not be connected or functioning.

Return Value

The function returns the number of auxiliary audio devices installed on the system.

Visual Basic-Specific Issues

None.

Parameters

None.

Example

' This code is licensed according to the terms and conditions listed here.
Loop through each auxiliary audio device installed on the system and display its name and version number.

```vba
dim auxinfo as auxcaps ' receives info about each device
dim numdevs as long ' number of auxiliary audio devices installed
dim devname as string ' name of device
dim majver as integer, minver as integer ' major and minor version numbers
dim c as long ' counter variable
dim retval as long ' return value

' Find out how many auxiliary audio devices are installed.
numdevs = auxgetnumdevs()

' Loop through each one, displaying its name and version number.
for c = 0 to numdevs - 1 ' remember that device IDs are zero-based!
    ' Get the capabilities of this device.
    retval = auxgetdevcaps(c, auxinfo, len(auxinfo))
    if retval = 0 then
        debug.print "** auxiliary audio device"; c; "**"
        ' Extract and display the name of the device.
        devname = left(auxinfo.szPname, instr(auxinfo.szPname, vbNullChar) - 1)
        debug.print "name: "; devname
        ' Extract and display the version number of the device.
        majver = (auxinfo.vDriverVersion and &hff00) / &h100 ' major version
        minver = auxinfo.vDriverVersion and &hff ' minor version
        debug.print "version number:"; majver; "."; minver
    else
        debug.print "Could not get information about device"; c; "."
    end if
next c
```

**Category**

Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** September 6, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
auxGetVolume Function

Declare Function auxGetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, lpdwVolume As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

auxGetVolume retrieves the current volume setting for an auxiliary audio device. This function will retrieve the volume whether the device supports dual-channel volume control or not.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

uDeviceID

The device ID of the auxiliary audio device to retrieve the volume of. Valid values range from 0 to the number of auxiliary audio devices minus one.

lpdwVolume

Receives the volume settings of the device. If the device supports separate left and right channel...
volumes, the low-order word contains the left channel volume and the high-order word contains the right channel volume. If the device does not support separate volumes, the low-order word contains the overall volume. Valid volume settings range from &H0 or &HFFFF.

**Example**

'D This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/a/auxgetvolume.html (2 of 3) [9/1/2002 5:03:45 PM]).

'Display the current volume setting of whatever device happens to be auxiliary audio device 0. This example identifies whether the device has dual-channel volumes or not.
Dim auxinfo As AUXCAPS ' receives information about the device
Dim numvols As Long ' identifies number of volumes on the device
Dim lrvol As Long ' volumes of both channels (or just the overall volume)
Dim lvol As Integer, rvol As Integer ' volumes of left and right channels
Dim retval As Long ' return value

' Figure out whether the device has one or two volume settings.
retval = auxGetDevCaps(0, auxinfo, Len(auxinfo))
If retval <> 0 Then ' error
    Debug.Print "Could not access auxiliary audio device 0 -- aborting."
    End ' give up
End If
If (auxinfo.dwSupport And AUXCAPS_LRVOLUME) = AUXCAPS_LRVOLUME Then
    numvols = 2 ' separate left and right volumes
Else
    numvols = 1 ' only one overall volume
End If

'Determine the device's current volume.
retval = auxGetVolume(0, lrvol)
'Display the current volume setting for the device.
If numvols = 2 Then
    ' Separate the left and right channel volumes. The next two lines look like an excessively complicated way of doing it, but because of a quirk in Visual Basic, the "obvious" way doesn't work properly.
lvol = Val("&H" & Hex(lrvol And (Not &HFFFF0000)))
rvol = (lrvol And &HFFFF0000) / &H10000
    ' Display the results in hexadecimal.
    Debug.Print "Left Channel volume: "; Hex(lvol)
    Debug.Print "Right Channel volume: "; Hex(rvol)
Else
    ' Extract the useful information as above, although we only want the low-order word (placed into lvol).

lvol = Val("&H" & Hex(lrvol And (Not &HFFFF0000)))

' Display the results in hexadecimal.
Debug.Print "Volume: "; hex(lvol)  ' here, lvol is the overall volume
End If

See Also

auxSetVolume

Category

Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 10, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/a/auxgetvolume.html
auxSetVolume Function

Declare Function auxSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

auxSetVolume sets the current volume of an auxiliary audio device. This function works whether the device supports separate left and right channel volumes or not.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

uDeviceID
The device ID of the auxiliary audio device to set the volume of. Valid values range from zero to the number of auxiliary audio devices minus one.

dwVolume
The new volume setting of the device. If the device supports separate left and right channel volumes,
the low-order word contains the left channel volume and the high-order word contains the right channel volume. If the device supports only one overall volume, the low-order word is the volume. Valid volumes range from &H0 to &HFFFF.

### Example

```
' This code is licensed according to the terms and conditions listed here.
' Set the volume of auxiliary audio device 0 to 50%. Here, ' we don't care whether the device supports separate volumes or ' not, since the code works either way.
Dim retval As Long  ' return value

' Set the volume of both channels (if a second exists) to 50% (&H8000).
retval = auxSetVolume(0, &H80008000)  ' both words are set
```

### See Also

auxGetVolume

### Category

Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** September 10, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/a/auxsetvolume.html
Beep Function

Declare Function Beep Lib "kernel32.dll" (ByVal dwFreq As Long, ByVal dwDuration As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

Beep plays a sound, but its exact behavior varies between platforms. **Windows 95/98**: The function always plays the SystemDefault system sound, regardless of the values passed to the function. **Windows NT/2000**: The function plays a tone through the computer's internal speaker at the desired frequency for a specified duration.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None

Parameters

- **dwFreq**
  - **Windows NT/2000**: The frequency, in hertz (Hz), of the tone to play. **Windows 95/98**: Ignored.
- **dwDuration**
  - **Windows NT/2000**: The duration, in milliseconds, to play the desired tone. **Windows 95/98**: Ignored.

Example
' This code is licensed according to the terms and conditions listed here.

' Attempt to play a note at 800 Hz for 2 seconds. This will only
' behave this way on Windows NT/2000; users of Windows 95/98 will only hear the
' default sound.
Dim retval As Long ' return value

retval = Beep(800, 2000) ' ideally, an 800 Hz tone for 2 seconds

See Also

MessageBeep

Category

Errors

Go back to the alphabetical Function listing.
Go back to the Reference section index.
BitBlt Function

Declare Function BitBlt Lib "gdi32.dll" (ByVal hdcDest As Long, ByVal nXDest As Long, ByVal nYDest As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hdcSrc As Long, ByVal nXSrc As Long, ByVal nYSrc As Long, ByVal dwRop As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

BitBlt performs a bit-block transfer of a rectangular portion of an image from one device to another. The dimensions of the transferred rectangle are perfectly preserved. The function can perform a variety of raster operations to transfer the block from the source device to the target device.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

**hdcDest**

A handle to the device context of the device which receives the transferred image block.
Windows API Guide: BitBlt Function

**nXDest**
The x-coordinate of the point to position the upper-left corner of the transferred image block.

**nYDest**
The y-coordinate of the point to position the upper-left corner of the transferred image block.

**nWidth**
The width in pixels of the image block.

**nHeight**
The height in pixels of the image block.

**hdcSrc**
A handle to the device context of the device which contains the image block to transfer.

**nXSrc**
The x-coordinate of the upper-left corner of the image block to transfer.

**nYSrc**
The y-coordinate of the upper-left corner of the image block to transfer.

**dwRop**
One of the following flags identifying the raster operation to use to transfer the image block. Each raster operation uses the RGB color value of the source and/or target pixel to determine the new color of the target pixel.

- **BLACKNESS**
  Fill the destination rectangle with the color whose index is 0 in the physical palette (which is black by default).

- **CAPTUREBLT**
  *Windows 98, 2000:*
  Include any windows layered on top of the window being used in the resulting image.

- **DSTINVERT**
  Invert the colors in the destination rectangle.

- **MERGECOPY**
  Merge the colors of the source rectangle with the specified pattern using the bitwise AND operator.

- **MERGEPAINT**
  Merge the colors of the inverted source rectangle with the colors of the destination rectangle using the bitwise OR operator.

- **NOMIRRORBITMAP**
  *Windows 98, 2000:*
  Prevent the bitmap from being mirrored.

- **NOTSRCOPY**
  Copy the inverted source rectangle to the destination rectangle.

- **NOTSRCERASE**
  Combine the colors of the source and destination rectangles using the bitwise OR operator and then invert the resulting color.

- **PATCOPY**
  Copy the specified pattern into the destination bitmap.

- **PATINVERT**
  Combine the colors of the specified pattern with the colors of the destination rectangle using the bitwise XOR operator.

- **PATPAINT**
  Combine the colors of the specified pattern with the colors of the inverted source rectangle using the bitwise OR operator. Combine the result of that operation with the colors of the destination rectangle using the bitwise OR operator.
SRCAND
Combine the colors of the source and destination rectangles using the bitwise AND operator.

SRCCOPY
Copy the source rectangle directly into the destination rectangle.

SRCERASE
Combine the inverted colors of the destination rectangle with the colors of the source rectangle using the bitwise AND operator.

SRCINVERT
Combine the colors of the source and destination rectangles using the bitwise XOR operator.

SRCPAINT
Combine the colors of the source and destination rectangles using the bitwise OR operator.

WHITENESS
Fill the destination rectangle with the color whose index is 1 in the physical palette (which is white by default).

Constant Definitions

Const BLACKNESS = &H42
' Const CAPTUREBLT = ???
Const DSTINVERT = &H550009
Const MERGECOPY = &HC000CA
Const MERGEPAINT = &HBB0226
' Const NOMIRRORBMAP = ???
Const NOTSRCCOPY = &H330008
Const NOTSRCERASE = &H1100A6
Const PATCOPY = &HF00021
Const PATINVERT = &H5A0049
Const PATPAINT = &HFB0A09
Const SRCAND = &H8800C6
Const SRCCOPY = &HCC0020
Const SRCERASE = &H440328
Const SRCINVERT = &H660046
Const SRCPAINT = &HEE0086
Const WHITENESS = &HFF0062

Example

' This code is licensed according to the terms and conditions listed here.

' Copy a rectangular image from window Form1 to window Form2
' exactly (using SRCCOPY). The rectangle has a width of 100 and a height of
' 50. The upper-left corner of the source block is (350, 250); the block
' is placed at (0,0) in Form2.
Dim retval As Long ' return value
' Transfer the image exactly as described above.
retval = BitBlt(Form2.hDC, 0, 0, 100, 50, Form1.hDC, 350, 250, SRCCOPY)

See Also

StretchBlt

Category

Bitmaps

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified:  August 13, 1999
This page is copyright © 1999 Paul Kuliniewicz.  Copyright Information  Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/b/bitblt.html
BringWindowToTop Function

Declare Function BringWindowToTop Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

BringWindowToTop brings a specified window to the top of the Z-order, placing it above any windows previously on top of it. This function has the same effect as using SetWindowPos to place the window at the top of the Z-order.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None

Parameters

hwnd

A handle to the window to bring to the top of the Z-order.

Example

' This code is licensed according to the terms and conditions listed here.
' Bring the window Form1 to the top of the Z-order.
Dim retval As Long  ' return value

' Obviously, this will only do something if other windows are already on top of Form1.
retval = BringWindowToTop(Form1.hWnd)

See Also

SetWindowPos

Category:

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

Last Modified: July 28, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/b/bringwindowtotop.html
CallWindowProc Function

Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CallWindowProc explicitly calls the hook function acting as a window's procedure to process a message. This allows a message for a window to be processed by a window procedure which is not necessarily the one normally called by the window.

Return Value

The function returns the return value generated after processing the message sent.

Visual Basic-Specific Issues

None.

Parameters

- lpPrevWndFunc
  A pointer to the window procedure function to call explicitly. This is the function which will process the message.
- hWnd
  A handle to the window to process the message for.
- Msg
  The message to process.
- wParam
  Additional information about the message.
- lParam
Example

' This code is licensed according to the terms and conditions listed here.

' Have window Form1 play the SystemAsterisk sound whenever it gets
' or loses the focus. Do this by specifying a new window procedure which
' plays the sound whenever the WM_ACTIVATE message is received. To
' process all other messages (and do whatever else WM_ACTIVATE should
' do), the procedure then calls the previous window procedure.

' *** Place this code in a module. ***
Const WM_ACTIVATE = &H6  ' identifier of the message
' The following variable is accessible to all code in this example.
Public pOldProc As Long  ' pointer to the previous window function

' Define the new window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Dim retval As Long
' If the message is WM_ACTIVATE (we don't care about the parameters),
' play the SystemAsterisk sound.
If uMsg = WM_ACTIVATE Then
    retval = PlaySound("SystemAsterisk", 0, SND_ALIAS Or SND_ASYNC)
End If
' No matter what happened, use the old window procedure to
' finish processing the message.
    retval = CallWindowProc(pOldProc, hWnd, uMsg, wParam, lParam)
' Have this function return whatever the function above returned.
WindowProc = retval
End Function

' *** Place the following code wherever you wish. ***
Dim retval As Long  ' return value

' Set the new window procedure for Form1, saving a pointer to the old one.
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
' Now WindowProc processes Form1's messages, playing the sound
' whenever Form1 is activated or loses activated status.

See Also

DefWindowProc

Category

Window Procedures
ChangeDisplaySettings Function

Declare Function ChangeDisplaySettings Lib "user32.dll" Alias "ChangeDisplaySettingsA" (lpDevMode As Any, ByVal dwFlags As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.5 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The `ChangeDisplaySettings` function changes the current display settings. This function can change the current screen resolution and color depth, among other things. Typically, a call to `EnumDisplaySettings` should precede this function, in order to adjust only the desired settings.

Return Value

The function returns one of the following flags:

- **DISP_CHANGE_SUCCESSFUL**: The display settings were successfully changed.
- **DISP_CHANGE_RESTART**: The computer must be restarted for the display changes to take effect.
- **DISP_CHANGE_BADFLAGS**: An invalid set of flags was specified.
- **DISP_CHANGE_BADPARAM**: An invalid parameter was specified.
- **DISP_CHANGE_FAILED**: The display driver failed the specified graphics mode.
- **DISP_CHANGE_BADMODE**: The specified graphics mode is not supported.
- **DISP_CHANGE_NOTUPDATED**: Windows NT/2000: The settings could not be written to the registry.

Visual Basic-Specific Issues
When specifying zero for *lpDevMode*, use the expression ByVal CLng(0).

**Parameters**

*lpDevMode*

A [DEVMODE](#) structure that holds the new display settings. Only the *dmBitsPerPixel*, *dmPelsWidth*, *dmPelsHeight*, *dmDisplayFlags*, and *dmDisplayFrequency* members are used. ([Windows 98, NT/2000]: The *dmPosition* member can also be used.) To restore the settings saved in the registry, set this parameter and *dwFlags* to zero.

*dwFlags*

A combination of the following flags specifying how to change the display mode. If no flags are set (i.e., zero is specified), the graphics mode is changed dynamically.

- **CDS_UPDATEREGISTRY**
  - Save the new settings to the registry and also change the settings dynamically.

- **CDS_TEST**
  - Test to see if the new settings are supported by the hardware, without actually changing the settings. The function's return value will indicate any problems that may have occurred.

- **CDS_FULLSCREEN**
  - Go into full-screen mode. This setting cannot be saved.

- **CDS_GLOBAL**
  - Save the new settings for all users. The CDS_UPDATEREGISTRY flag must also be specified.

- **CDS_SET_PRIMARY**
  - Make this device the primary display device.

- **CDS_RESET**
  - Change the settings even if they are the same as the current settings.

- **CDS_NORESET**
  - Save the settings to the registry, but do not make them take effect yet. The CDS_UPDATEREGISTRY flag must also be specified.

**Constant Definitions**

```vbnet
Const CDS_UPDATEREGISTRY = &H1
Const CDS_TEST = &H2
Const CDS_FULLSCREEN = &H4
Const CDS_GLOBAL = &H8
Const CDS_SET_PRIMARY = &H10
Const CDS_RESET = &H40000000
Const CDS_SETRECT = &H20000000
Const CDS_NORESET = &H10000000
Const DISP_CHANGE_SUCCESSFUL = 0
Const DISP_CHANGE_RESTART = 1
Const DISP_CHANGE_FAILED = -1
Const DISP_CHANGE_BADMODE = -2
Const DISP_CHANGE_NOTUPDATED = -3
Const DISP_CHANGE_BADFLAGS = -4
Const DISP_CHANGE_BADPARAM = -5
```
Example

Change the current screen resolution to 800x600 without changing the color depth and save the changes to the registry. First test to make sure that the hardware supports the new resolution. If a reboot is necessary, inform the user. To use this example, place a command button named Command1 of a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type DEVMODE
    dmDeviceName As String * 32
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    dmFormName As String * 32
    dmUnusedPadding As Integer
    dmBitsPerPixel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long
' The following only appear in Windows 95, 98, 2000
    dmICMMethod As Long
    dmICMIntent As Long
    dmMediaType As Long
    dmDitherType As Long
    dmReserved1 As Long
    dmReserved2 As Long
' The following only appear in Windows 2000
    dmPanningWidth As Long
    dmPanningHeight As Long
End Type
Public Declare Function EnumDisplaySettings Lib "user32.dll" Alias

"EnumDisplaySettingsA" (ByVal lpszDeviceName As String, _
   ByVal iModeNum As Long, lpDevMode As DEVMODE) As Long
Public Const ENUM_CURRENT_SETTINGS = -1
Public Declare Function ChangeDisplaySettings Lib "user32.dll" Alias
   "ChangeDisplaySettingsA" (lpDevMode As Any, ByVal dwFlags _
   As Long) As Long
Public Const CDS_UPDATEREGISTRY = &H1
Public Const CDS_TEST = &H2
Public Const DISP_CHANGE_SUCCESSFUL = 0
Public Const DISP_CHANGE_RESTART = 1

' *** Place the following code inside the form window.

Private Sub Command1_Click()
   Dim dm As DEVMODE   ' display settings
   Dim retval As Long  ' return value
   ' Initialize the structure that will hold the settings.
   dm.dmSize = Len(dm)
   ' Get the current display settings.
   retval = EnumDisplaySettings(vbNullString, ENUM_CURRENT_SETTINGS, dm)
   ' Change the resolution settings to 800x600.
   dm.dmPelsWidth = 800
   dm.dmPelsHeight = 600
   ' Test to make sure the changes are possible.
   retval = ChangeDisplaySettings(dm, CDS_TEST)
   If retval <> DISP_CHANGE_SUCCESSFUL Then
      Debug.Print "Cannot change to that resolution!"
   Else
      ' Change and save to the new settings.
      retval = ChangeDisplaySettings(dm, CDS_UPDATEREGISTRY)
      Select Case retval
      Case DISP_CHANGE_SUCCESSFUL
         Debug.Print "Resolution successfully changed!"
      Case DISP_CHANGE_RESTART
         Debug.Print "A reboot is necessary before the changes will take effect."
      Case Else
         Debug.Print "Unable to change resolution!"
      End Select
   End If
End Sub

See Also

EnumDisplaySettings

Category
CharLower Function

Declare Function CharLower Lib "user32.dll" Alias "CharLowerA" (ByVal lpsz As String) As String

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CharLower converts all of the upper-case letters in a string into lower-case. The function also sets the string passed to the function to the returned string; in reality they become the same thing.

Return Value

The function returns the string, with all upper-case letters converted to lower-case.

Visual Basic-Specific Issues

None.

Parameters

lpsz

The string to convert to lower-case.

Example
' This code is licensed according to the terms and conditions listed here.

' Convert the string "Hello, World!" into lower-case.

Dim target As String  ' target string

target = CharLower("Hello, World!")  ' Convert to lower-case
Debug.Print target  ' Output should be "hello, world!"

**See Also**

CharUpper

**Category**

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** July 30, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/charlower.html
CharUpper Function

Declare Function CharUpper Lib "user32.dll" Alias "CharUpperA" (ByVal lpsz As String) As String

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CharUpper converts all of the lower-case letters in a string into upper-case. The function also sets the string passed to the function to the returned string; in reality they become the same thing.

Return Value

The function returns the string, with all lower-case letters converted to upper-case.

Visual Basic-Specific Issues

None.

Parameters

lpsz

The string to convert to upper-case.

Example
' This code is licensed according to the terms and conditions listed here.

' Convert the string "Hello, World!" into upper-case.

Dim target As String  ' target string
target = CharUpper("Hello, World!")  ' Convert to upper-case
Debug.Print target  ' Output should be "HELLO, WORLD!"

See Also

CharLower

Category

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

Last Modified: July 30, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/charupper.html
ChooseColor Function

Declare Function ChooseColor Lib "comdlg32.dll" Alias "ChooseColorA" (lpcc As CHOOSECOLOR_TYPE) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

ChooseColor opens a Choose Color common dialog box. All the information needed to create the dialog box, as well as the data returned from it, is stored in the structure passed as lpcc.

Return Value

If an error occurred or the user pressed the Cancel button, the function returns 0 (use CommDlgExtendedError to get the error code). If the user successfully selected a color, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpcc

All the information needed to create the Choose Color common dialog box. If successful, the function then stores the user's color selection and list of custom colors into this structure as well.

Example

' This code is licensed according to the terms and conditions listed here.

' Display a Choose Color common dialog box. The background
Windows API Guide: ChooseColor Function

' color of Form1 will be set to the color the user selects. Although the entire
' list of custom colors is initialized to black, this example stores the
' colors into an array which can be used again to save the user's custom
' color selections.
Dim cc As CHOOSECOLOR_TYPE ' structure to pass data
Dim hMem As Long ' handle to the memory block to store the custom color list
Dim pMem As Long ' pointer to the memory block to store the custom color list
Dim custcols(0 To 15) As Long ' holds list of the 16 custom colors
Dim c As Integer ' counter variable
Dim retval As Long ' return value
' Initialize the list of custom colors to black.
For c = 0 To 15 ' loop through each element
custcols(c) = 0 ' set each element to RGB color 0 (black)
Next c
' Create a memory block and get a pointer to it.
hMem = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, 64) ' allocate sufficient memory
block
pMem = GlobalLock(hMem) ' get a pointer to the block
' Copy the data inside the array into the memory block.
CopyMemory ByVal pMem, custcols(0), 64 ' 16 elements * 4 bytes
' Store the initial settings of the Choose Color box.
cc.lStructSize = Len(cc) ' size of the structure
cc.hwndOwner = Form1.hWnd ' Form1 is opening the Choose Color box
cc.hInstance = 0 ' not needed
cc.rgbResult = Form1.BackColor ' set default selected color to Form1's background
color
cc.lpCustColors = pMem ' pointer to list of custom colors
cc.Flags = CC_ANYCOLOR Or CC_RGBINIT ' allow any color, use rgbResult as default
selection
cc.lCustData = 0 ' not needed
cc.lpfnHook = 0 ' not needed
cc.lpTemplateName = "" ' not needed
' Open the Choose Color box. If the user chooses a color, set Form1's
' background color to that color.
retval = ChooseColor(cc)
If retval <> 0 Then ' success
' Copy the possibly altered contents of the custom color list
' back into the array.
CopyMemory custcols(0), ByVal pMem, 64
' Set Form1's background color.
Form1.BackColor = cc.rgbResult
End If
' Deallocate the memory blocks to free up resources.
retval = GlobalUnlock(hMem)
retval = GlobalFree(pMem)



ChooseColor Function

Category

Common Dialog

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

Last Modified: August 14, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/choosecolor.html
ChooseFont Function

Declare Function ChooseFont Lib "comdlg32.dll" Alias "ChooseFontA" (lpcf As CHOOSEFONT_TYPE) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ChooseFont opens the Choose Font common dialog box. All of the necessary information needed to create the dialog box, as well as all of the information returned from it, is stored in the structure passed as pChooseFont. The logical font information is mostly stored in a LOGFONT structure, which can be used to access that font.

Return Value

If an error occurred or the user pressed Cancel, the function returns 0 (use CommDlgExtendedError to get the error code). If the user successfully selected a font, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpcf

Passes information to and from the Choose Font dialog box. Initialize this before calling the function, and read the necessary information from it afterwards.

Example

' This code is licensed according to the terms and conditions listed here.
'Display a Choose Font dialog box. Print out the typeface name, point size, and style of the selected font. More detail about topics in this example can be found in the pages for **CHOOSEFONT_TYPE** and **LOGFONT**.

Dim cf As **CHOOSEFONT_TYPE** ' data structure needed for function
Dim lfont As **LOGFONT** ' receives information about the chosen font
Dim hMem As Long, pMem As Long ' handle and pointer to memory buffer
Dim fontName As String ' receives name of font selected
Dim retval As Long ' return value

' Initialize the default selected font: Times New Roman, regular, black, 12 point. (Note that some of that information is in the CHOOSEFONT_TYPE structure instead.)
lfont.lfHeight = 0 ' determine default height
lfont.lfWidth = 0 ' determine default width
lfont.lfEscapement = 0 ' angle between baseline and escapement vector
lfont.lfOrientation = 0 ' angle between baseline and orientation vector
lfont.lfWeight = FW_NORMAL ' normal weight i.e. not bold
lfont.lfItalic = 0 ' not italic
lfont.lfUnderline = 0 ' not underline
lfont.lfStrikeOut = 0 ' not strikeout
lfont.lfCharSet = DEFAULT_CHARSET ' use default character set
lfont.lfOutPrecision = OUT_DEFAULT_PRECIS ' default precision mapping
lfont.lfClipPrecision = CLIP_DEFAULT_PRECIS ' default clipping precision
lfont.lfQuality = DEFAULT_QUALITY ' default quality setting
lfont.lfPitchAndFamily = DEFAULT_PITCH Or FF_ROMAN ' default pitch, proportional with serifs
lfont.lfFaceName = "Times New Roman" & vbNullChar ' string must be null-terminated

' Create the memory block which will act as the LOGFONT structure buffer.
hMem = **GlobalAlloc**(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(lfont))
pMem = **GlobalLock**(hMem) ' lock and get pointer
**CopyMemory** ByVal pMem, lfont, Len(lfont) ' copy structure's contents into block

' Initialize dialog box: Screen and printer fonts, point size between 10 and 72.
cf.lStructSize = Len(cf) ' size of structure
cf.hwndOwner = Form1.hWnd ' window Form1 is opening this dialog box
cf.hdc = Printer.hDC ' device context of default printer (using VB's mechanism)
cf.lfLogFont = pMem ' pointer to LOGFONT memory block buffer
cf.iPointSize = 120 ' 12 point font (in units of 1/10 point)
cf.flags = CF_BOTH Or CF_EFFECTS Or CF_FORCEFONTEXIST Or CF_INITTOLOGFONTSTRUCT Or CF_LIMITSIZE
cf.rgbColors = RGB(0, 0, 0) ' black
cf.lpfnData = 0 ' we don't use this here...
cf.lpfnHook = 0 ' ...or this...
cf.lpTemplateName = "" ' ...or this...
cf.hInstance = 0 ' ...or this...
cf.lpszStyle = "" ' ...or this
 cf.nFontType = REGULAR_FONTTYPE ' regular font type i.e. not bold or anything
cf.nSizeMin = 10 ' minimum point size
cf.nSizeMax = 72 ' maximum point size
' Now, call the function. If successful, copy the LOGFONT structure back into the structure
' and then print out the attributes we mentioned earlier that the user selected.
retval = ChooseFont(cf) ' open the dialog box
If retval <> 0 Then ' success
  CopyMemory lfont, ByVal pMem, Len(lfont) ' copy memory back
' Now make the fixed-length string holding the font name into a "normal" string.
  fontname = Left(lfont.lfFaceName, InStr(lfont.lfFaceName, vbNullChar) - 1)
' Display font name and a few attributes.
  Debug.Print "FONT NAME: "; fontname
  Debug.Print "FONT SIZE (points):"; cf.iPointSize / 10 ' in units of 1/10 point!
  Debug.Print "FONT STYLE(S): ";
    If lfont.lfWeight >= FW_BOLD Then Debug.Print "Bold ";
    If lfont.lfItalic <> 0 Then Debug.Print "Italic ";
    If lfont.lfUnderline <> 0 Then Debug.Print "Underline ";
    If lfont.lfStrikeOut <> 0 Then Debug.Print "Strikeout";
  Debug.Print ' end the line
End If
'
' Deallocate the memory block we created earlier. Note that this must
' be done whether the function succeeded or not.
retval = GlobalUnlock(hMem) ' destroy pointer, unlock block
retval = GlobalFree(hMem) ' free the allocated memory

Category

Common Dialog

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 19, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/choosefont.html
Chord Function

Declare Function Chord Lib "gdi32.dll" (ByVal hdc As Long, ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long, ByVal nXRadial1 As Long, ByVal nYRadial1 As Long, ByVal nXRadial2 As Long, ByVal nYRadial2 As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

Chord draws an elliptical chord on a device. The chord is drawn using the device's currently selected pen and is filled using its currently selected brush. The chord consists of a line segment connecting two points along an ellipse; the area between the chord and the ellipse's edge is filled (going counterclockwise around the ellipse). The start and end points of the chord's elliptical arc are determined by two radials (drawn from the ellipse's center out to a point). The point where a radial and the ellipse intersect is the start or end point of the chord's arc.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- hdc: A handle to a device context to the device to draw the chord on.
- nLeftRect, nTopRect, nRightRect, nBottomRect, nXRadial1, nYRadial1, nXRadial2, nYRadial2: Locations and radials used to draw the chord.
The x-coordinate of the upper-left corner of the ellipse's bounding rectangle.

$nTopRect$
The y-coordinate of the upper-left corner of the ellipse's bounding rectangle.

$nRightRect$
The x-coordinate of the lower-right corner of the ellipse's bounding rectangle.

$nBottomRect$
The y-coordinate of the lower-right corner of the ellipse's bounding rectangle.

$nXRadial1$
The x-coordinate of the point along the radial determining the starting point of the chord.

$nYRadial1$
The y-coordinate of the point along the radial determining the starting point of the chord.

$nXRadial2$
The x-coordinate of the point along the radial determining the ending point of the chord.

$nYRadial2$
The y-coordinate of the point along the radial determining the ending point of the chord.

Example

' This code is licensed according to the terms and conditions listed here.

' Draw a chord on window Form1. The ellipse has a bounding rectangle
' of (10,20)-(210,120). The chord will have endpoints on the ellipse of (210,70)
' and (110,20) -- i.e., the "upper-right" portion of the ellipse. Draw the chord using
' Form1's current brush and pen.
Dim retval As Long  ' return value

' Draw the chord as specified above.
retval = Chord(Form1.hDC, 10, 20, 210, 120, 210, 70, 110, 20)

See Also

Ellipse, Pie

Category

Filled Shapes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 10, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/chord.html
ClipCursor Function

Declare Function ClipCursor Lib "user32.dll" (lpRect As RECT) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ClipCursor confines the mouse cursor to a rectangular area of the screen. If the user tries to move the cursor outside of this bounding region or a call to SetCursorPos tells it to go outside the box, the cursor will immediately returned to the area. There is no way to get it out. This bounding effect will last in whatever program you switch to, and will remain even if the program that confined the cursor closes! The only way to "release" the cursor is to "confine" it to the entire screen (see example). It isn't usually a good idea to confine the cursor, since the user expects to move the cursor anywhere (not to mention the disastrous effect if your program quit before releasing the cursor!).

Return Value

If an error occurred, the function returns zero (call GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters
**lpRect**

The rectangle (in screen coordinates) defining the confinement rectangle.

### Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/c/clipcursor.html).

' Confine the cursor temporarily to inside of Form1

' ** Place the following code where you want to confine the cursor. **
Dim r As RECT  ' confinement rectangle
Dim retval As Long  ' return value

retval = GetWindowRect(Form1.hWnd, r)  ' put window's coordinates into r
retval = ClipCursor(r)  ' confine the cursor

' ** Place the following code where you want to release the cursor. **
Dim r As RECT, retval As Long
Dim desk hWnd As Long  ' the handle of the desktop window

desk hWnd = GetDesktopWindow()  ' get handle of the desktop window
retval = GetWindowRect(desk hWnd, r)  ' put window's coordinates into r
retval = ClipCursor(r)  ' "confine" the cursor to the entire screen

### See Also

GetClipCursor

### Category

Cursor


CloseHandle Function

Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CloseHandle closes a handle and the object associated with that handle. After being closed, the handle is of course no longer valid. This function closes handles associated with access tokens, communications devices, console inputs, console screen buffers, events, files, file mappings, jobs, mailslots, mutexes, named pipes, processes, semaphores, sockets, and threads.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

hObject

A handle to the object to close.

Example

' This code is licensed according to the terms and conditions listed here.

' Display the date on which the file C:\MyApp\test.txt was created. Note how the time zone conversion is necessary.
Dim hFile As Long ' handle to the opened file
Dim ctime As FILETIME  ' receives time of creation
Dim atime As FILETIME  ' receives time of last access
Dim mtime As FILETIME  ' receives time of last modification
Dim thetime As SYSTEMTIME  ' used to manipulate the time
Dim retval As Long  ' return value

' First, open the file C:\MyApp\test.txt for read-level access.  Note the
' expression necessary to pass 0 as lpSecurityAttributes.
hFile = CreateFile("C:\MyApp\test.txt", GENERIC_READ, FILE_SHARE_READ, ByVal CLng(0),
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hFile = -1 Then
    Debug.Print "Could not open the file successfully -- aborting."
    End ' terminate the program
End If

' Next, get the creation, last-access, and last-modification times.
retval = GetFileTime(hFile, ctime, atime, mtime)
' Convert the creation time to the local time zone.
retval = FileTimeToLocalFileTime(ctime, ctime)
' Convert the FILETIME format to the SYSTEMTIME format.
retval = FileTimeToSystemTime(ctime, thetime)

' Display the date of creation of the file to the user.
Debug.Print "The file was created on "; thetime.wMonth; "-"; thetime.wDay; "-";
          thetime.wYear

' Close the file to free up resources.
retval = CloseHandle(hFile)

Category

Handles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
ClosePrinter Function

Declare Function ClosePrinter Lib "winspool.drv" (ByVal hPrinter As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ClosePrinter closes a printer which had been previously opened by OpenPrinter. After this function closes the printer, the handle to the printer obviously can no longer be used, even if the function fails.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hPrinter
A handle to the open printer to close.

Example

Display the Properties dialog box for the system's default printer. The dialog box is opened when the user clicks the button cmdProperties. To use the example, first place a command button named cmdProperties on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
Public Type **PRINTER_INFO_1**
    flags As Long
    pDescription As String
    pName As String
    pComment As String
End Type

Public Declare Function **EnumPrinters** Lib "winspool.drv" Alias "EnumPrintersA" (ByVal flags As Long, ByVal name As String, ByVal Level As Long, pPrinterEnum As Long, ByVal cdBuf As Long, pcbNeeded As Long, pcReturned As Long) As Long
Public Const PRINTER_ENUM_DEFAULT = &H1

Public Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long

Public Declare Function **OpenPrinter** Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, pDefault As String) As Long

Public Declare Function **PrinterProperties** Lib "winspool.drv" (ByVal hWnd As Long, ByVal hPrinter As Long) As Long

Public Declare Function **ClosePrinter** Lib "winspool.drv" (ByVal hPrinter As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub cmdProperties_Click()
    Dim pil As **PRINTER_INFO_1**  ' a little info about the printer
    Dim bytesNeeded As Long      ' size needed for buffer
    Dim numPrinters As Long      ' number of printers enumerated (should be 1)
    Dim buffer() As Long        ' buffer for printer information
    Dim slength As Long         ' length of string to copy
    Dim hPrinter As Long        ' handle to the printer
    Dim retval As Long          ' generic return value

    ' Figure out how much space is needed to store the printer information.
    retval = **EnumPrinters** (PRINTER_ENUM_DEFAULT, vbNullString, 1, ByVal 0, 0, bytesNeeded, numPrinters)
    ' Allocate that much space in the buffer array.
    ReDim buffer(0 To bytesNeeded / 4 - 1) As Long
    ' Get information about the default printer.
    retval = **EnumPrinters** (PRINTER_ENUM_DEFAULT, vbNullString, 1, buffer(0), bytesNeeded, _
                                bytesNeeded, numPrinters)
    ' Make sure we were successful.
    If retval = 0 Or numPrinters <> 1 Then
        Debug.Print "No default printer or some other error."
        Exit Sub
    End If
' Copy the data into the structure.
With pi1
    ' Copy numerical data directly.
    .flags = buffer(0)
' Strings require more work, since the buffer holds pointers to them.
    .pDescription = Space(lstrlen(buffer(1)))
    retval = lstrcpy(.pDescription, buffer(1))
    .pName = Space(lstrlen(buffer(2)))
    retval = lstrcpy(.pName, buffer(2))
    .pComment = Space(lstrlen(buffer(3)))
    retval = lstrcpy(.pComment, buffer(3))
End With

' Open the printer.
retval = OpenPrinter(pi1.pName, hPrinter, ByVal CLng(0))
If retval <> 0 Then
    ' Display the properties dialog.
    retval = PrinterProperties(Me.hWnd, hPrinter)
    ' Close the printer.
    retval = ClosePrinter(hPrinter)
Else
    Debug.Print "Unable to open printer!"
End If
End Sub

See Also

OpenPrinter

Category

Printers

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/closeprinter.html
closesocket Function

Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

closesocket closes a socket that had previously been created by a call to socket. Your program should call this function when it is finished using a socket, in order to free unneeded socket descriptors.

Return Value

If successful, the function returns zero. If an error occurred, the function returns INVALID_SOCKET (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

s

A descriptor of the socket to close.

Constant Definitions

Const INVALID_SOCKET = &HFFFFFFFE

Example

Download the main page of this web site (http://www.vbapi.com). This example supports a very crude implementation of
HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer
Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long
Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type
Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long
Declare Function `ioctlsocket` Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long
Public Const FIONBIO = &H8004667E
Public Declare Function `send` Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function `recv` Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function `closesocket` Lib "wsock32.dll" (ByVal s As Long) As Long
Public Declare Sub `CopyMemory` Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Const SOCKET_ERROR = -1

' Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA    ' info about Winsock
    Dim sock As Long            ' the socket descriptor
    Dim pHostinfo As Long       ' pointer to info about the host computer
    Dim hostinfo As hostent     ' info about the host computer
    Dim pIPAddress As Long      ' pointer to host's IP address
    Dim ipAddress As Long       ' host's IP address
    Dim sockinfo As sockaddr    ' settings for the socket
    Dim buffer As String        ' buffer for sending and receiving data
    Dim reply As String         ' accumulates server's reply
    Dim retval As Long          ' generic return value

    ' Begin a Winsock session.
    retval = WSAStartup(MAKEWORD(2, 2), wsockinfo)
    If retval <> 0 Then
        Debug.Print "Unable to initialize Winsock! --"; retval
        Exit Sub
    End If

    ' Get information about the server to connect to.
    pHostinfo = gethostbyname("www.vbapi.com")
    If pHostinfo = 0 Then
        Debug.Print "Unable to resolve host!"
        GoTo Cleanup
    End If

    ' Copy information about the server into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print " Couldn't get IP address of www.vbapi.com!"
    End If

GoTo Cleanup

' Get the server's IP address out of the structure.
CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4

' Create a socket.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If

' Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
    ' Use Internet Protocol (IP)
    .sin_family = AF_INET
    ' Connect to port 80.
    .sin_port = htons(80)
    ' Connect to this IP address.
    .sin_addr = ipAddress
    ' Padding characters.
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If

' Send an HTTP/GET request for the /index.html file.
buffer = "GET / HTTP/1.1" & vbCrLf & _
         "Host: www.vbapi.com" & vbCrLf & _
         "User-Agent: HTTP-Test-Program" & vbCrLf & vbCrLf
retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request.  Waiting for reply..."

' Make the socket non-blocking, so calls to recv don't halt the program waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)

' Read the response from the other system.  A more sophisticated program would watch to see if the connection ever times out (i.e., if the connection is lost).  For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If

' Process background events so the program doesn't appear to freeze.
DoEvents
Loop Until retval = 0

' Print the response from the server.
Debug.Print "Document Retrieved:"
Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()
End Sub

See Also

socket

Category

Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/closesocket.html
CombineRgn Function

Declare Function CombineRgn Lib "gdi32.dll" (ByVal hDestRgn As Long, ByVal hSrcRgn1 As Long, ByVal hSrcRgn2 As Long, ByVal nCombineMode As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**CombineRgn** combines two regions to form a third region. The two regions can be combined using a variety of logical operators. Note that the region that receives the combined regions must already be a region -- the function cannot create a new one but can change an existing one.

Return Value

The function returns one of the following flags specifying the result of the region combination operation:

- **ERROR**: An error occurred while trying to combine the regions.
- **NULLREGION**: The combined region is empty, i.e., null.
- **SIMPLEREGION**: The combined region forms a rectangle.
- **COMPLEXREGION**: The combined region is not empty but is also not a rectangle.

Visual Basic-Specific Issues

None.

Parameters

**hDestRgn**

A handle to the region to be set to the combination of the two source regions. This region must already have been
created, although its contents when passed to the function is irrelevant.

\( h_{SrcRgn1} \)

The first of the two source regions.

\( h_{SrcRgn2} \)

The second of the two source regions.

\( n_{CombineMode} \)

One of the following flags specifying the logical operation to use to combine the two regions:

- RGN_AND
  The combined region is the overlapping area of the two source regions.

- RGN_OR
  The combined region is all the area contained in either of the two source regions, including any overlap.

- RGN_XOR
  The combined region is all of the area contained in either of the two source regions, excluding any overlap.

- RGN_DIFF
  The combined region is all the area of the first source region except for the portion also included in the second source region.

- RGN_COPY
  The combined region is identical to the first source region. The second source region is ignored.

### Constant Definitions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>0</td>
</tr>
<tr>
<td>NULLREGION</td>
<td>1</td>
</tr>
<tr>
<td>SIMPLEREGION</td>
<td>2</td>
</tr>
<tr>
<td>COMPLEXREGION</td>
<td>3</td>
</tr>
<tr>
<td>RGN_AND</td>
<td>1</td>
</tr>
<tr>
<td>RGN_OR</td>
<td>2</td>
</tr>
<tr>
<td>RGN_XOR</td>
<td>3</td>
</tr>
<tr>
<td>RGN_DIFF</td>
<td>4</td>
</tr>
<tr>
<td>RGN_COPY</td>
<td>5</td>
</tr>
</tbody>
</table>

### Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/c/combinergn.html).

' On window Form1, create overlapping elliptical and rectangular regions.
' Fill the overlapped area with a dark gray brush; fill the nonoverlapping areas with a light gray brush.
Dim hRgn1 As Long, hRgn2 As Long ' elliptical and rectangular source regions.
Dim hXorRgn As Long, hAndRgn As Long ' regions set to the non-intersection and intersection
Dim hLightBrush As Long, hDarkBrush As Long ' handles to light gray and dark gray brushes
Dim retval As Long ' return value

' Create the four regions. The initial settings of hXorRgn and hAndRgn are irrelevant.
hRgn1 = [CreateEllipticRgn](100, 50, 200, 100) ' bounding rect (100,50)-(200,100)
hRgn2 = [CreateRectRgn](150, 75, 300, 200) ' rectangle (150,75)-(300,200)
hXorRgn = [CreateRectRgn](0, 0, 0, 0) ' meaningless initialization
hAndRgn = CreateRectRgn(0, 0, 0, 0) ' meaningless initialization

' Now set hAndRgn to the intersection of the two source regions and hXorRgn to
' the non-intersection of the two source regions.
retval = CombineRgn(hXorRgn, hRgn1, hRgn2, RGN_XOR) ' non-intersection
retval = CombineRgn(hAndRgn, hRgn1, hRgn2, RGN_AND) ' intersection

' Now get the necessary stock brushes and fill in the two combined regions.
hLightBrush = GetStockObject(LTGRAY_BRUSH) ' light gray solid brush
hDarkBrush = GetStockObject(DKGRAY_BRUSH) ' dark gray solid brush
retval = FillRgn(Form1.hDC, hXorRgn, hLightBrush) ' fill non-intersection
retval = FillRgn(Form1.hDC, hAndRgn, hDarkBrush) ' fill intersection

' Delete the four regions to free up resources.
retval = DeleteObject(hRgn1)
retval = DeleteObject(hRgn2)
retval = DeleteObject(hXorRgn)
retval = DeleteObject(hAndRgn)
CommDlgExtendedError Function

Declare Function CommDlgExtendedError Lib "comdlg32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

CommDlgExtendedError returns the error code from the last common dialog box function. This function does not return error codes for any other API function; for that, use GetLastError instead. The function's return value is undefined if the last common dialog function call was successful. If an error with a common dialog function did occur, the return value is exactly one of the following common dialog error flags:

CDERR_DIALOGFAILURE = &HFFFF
The function could not open the dialog box.

CDERR_FINDRESFAILURE = &H6
The function failed to find the desired resource.

CDERR_GENERALCODES = &H0
The error involved a general common dialog box property.

CDERR_INITIALIZATION = &H2
The function failed during initialization (probably insufficient memory).

CDERR_LOADRESFAILURE = &H7
The function failed to load the desired resource.

CDERR_LOADSTRFAILURE = &H5
The function failed to load the desired string.

CDERR_LOCKRESFAILURE = &H8
The function failed to lock the desired resource.

CDERR_MEMALLOCFAILURE = &H9
The function failed to allocate sufficient memory.

CDERR_MEMLOCKFAILURE = &HA
The function failed to lock the desired memory.

CDERR_NOHINSTANCE = &H4
The function was not provided with a valid instance handle (if one was required).

CDERR_NOHOOK = &HB
The function was not provided with a valid hook function handle (if one was required).

CDERR_NOTEMPLATE = &H3
The function was not provided with a valid template (if one was required).

CDERR_REGISTERMSGFAIL = &HC
The function failed to successfully register a window message.

CDERR_STRUCTSIZE = &H1
The function was provided with an invalid structure size.

CFERR_CHOOSEFONTCODES = &H2000
The error involved the Choose Font common dialog box.

CFERR_MAXLESSTHANMIN = &H2002
The function was provided with a maximum font size value smaller than the provided minimum font size.

CFERR_NOFONTS = &H2001
The function could not find any existing fonts.

FNERR_BUFFERTOOSMALL = &H3003
The function was provided with a filename buffer which was too small.

FNERR_FILENAMECODES = &H3000
The error involved the Open File or Save File common dialog box.

FNERR_INVALIDFILENAME = &H3002
The function was provided with or received an invalid filename.

FNERR_SUBCLASSFAILURE = &H3001
The function had insufficient memory to subclass the list box.

FRERR_BUFFERLENGTHZERO = &H4001
The function was provided with an invalid buffer.

FRERR_FINDREPLACECODES = &H4000
The error involved the Find or Replace common dialog box.

PDERR_CREATEICFAILURE = &H100A
The function failed to create an information context.

PDERR_DEFAULTDIFFERENT = &H100C
The function was told that the information provided described the default printer, but the default printer's actual settings were different.

PDERR_DNDMMISMATCH = &H1009
The data in the two data structures describe different printers (i.e., they hold conflicting information).

PDERR_GETDEVMODEFAIL = &H1005
The printer driver failed to initialize the DEVMODE structure.

PDERR_INITFAILURE = &H1006
The function failed during initialization.

PDERR_LOADDRVFAILURE = &H1004
The function failed to load the desired device driver.

PDERR_NODEFAULTPRN = &H1008
The function could not find a default printer.

PDERR_NODEVICES = &H1007
The function could not find any printers.

PDERR_PARSEFAILURE = &H1002
The function failed to parse the printer-related strings in WIN.INI.

PDERR_PRINTERCODES = &H1000
The error involved the Print common dialog box.

PDERR_PRINTERNOTFOUND = &H100B
The function could not find information in WIN.INI about the requested printer.

PDERR_RETDEFFAILURE = &H1003
The handles to the data structures provided were nonzero even though the function was asked to return information about the default printer.

PDERR_SETUPFAILURE = &H1001
The function failed to load the desired resources.

Example:

' Give the Open File dialog box an insufficient buffer size.
' Then display the error code provided.
Dim filebox As OPENFILENAME  ' structure that sets the dialog box
Dim fname As String  ' will receive selected file's name
Dim retval As Long  ' return value
Dim errcode As long  ' receives the error code
' Configure how the dialog box will look
filebox.lStructSize = Len(filebox)  ' the size of the structure
filebox.hwndOwner = Form1.hWnd  ' handle of the form calling the function
filebox.lpstrTitle = "Open File"  ' text displayed in the box's title bar
' The next line sets up the file types drop-box
filebox.lpstrFilter = "Text Files" & vbNullChar & "*.txt" & vbNullChar & "All Files" & vbNullChar & ".*" & vbNullChar & vbNullChar
filebox.lpstrFile = ""  ' ERROR: an empty buffer!
filebox.nMaxFile = 0  ' length of file and pathname buffer
filebox.lpstrFileName = Space(255)  ' initialize buffer that receives filename of file
filebox.nMaxFileName = 255  ' length of filename buffer
' Allow only existing files and hide the read-only check box
filebox.flags = OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST Or OFN_HIDEREADONLY

' Execute the dialog box
retval = GetOpenFileName(filebox)
If retval = 0 Then  ' some error occurred, or Cancel was pressed
    errcode = CommDlgExtendedError()  ' get the error code for GetOpenFileName
    If errcode = FNERR_BUFFERTOOSMALL Then
        Debug.Print "The buffer provided was too small to hold the filename."
    If errcode = FNERR_INVALIDFILENAME Then
        Debug.Print "An invalid filename was provided."
    ' etc.
    End If
End If

See Also: GetLastError
Category: Common Dialog

Go back to the alphabetical Function listing.
Go back to the Reference section index.
**CompareFileTime Function**

Declare Function CompareFileTime Lib "kernel32.dll" (lpFileTime1 As FILETIME, lpFileTime2 As FILETIME) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

*CompareFileTime* compares two times stored in *FILETIME* format. The function determines which of the two times, if any, comes before the other chronologically. If the first time is earlier than the second time, the function returns -1. If the two times are equal, the function returns 0. If the first time is later than the second time, the function returns 1.

*lpFileTime1*
- The first of the two times to compare.

*lpFileTime2*
- The second of the two times to compare.

**Example:**

' Determine if file C:\MyProgram\datafile.txt was created before ' Jan 5, 1999.  Note how CreateFile's alternate declare must be used under Win 95/98 - ' see that function's page for more information.
Dim hfile As Long  ' receives the handle to the file
Dim ctime As FILETIME  ' receives creation date and time of the file
Dim atime As FILETIME  ' receives last access date and time of the file
Dim wtime As FILETIME  ' receives last write-to date and time of the file
Dim jantime As SYSTEMTIME  ' will be set to Jan 5, 1999
Dim janfiletime As FILETIME  ' will receive analogous time as jantime
Dim comptimes As Long  ' receives comparison of ctime and janfiletime
Dim retval As Long  ' return value

' Get a handle to the file (note how the alternate declare is used):
hfile = CreateFileNS("C:\MyProgram\datafile.txt", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then  ' if the file could not be opened
    Debug.Print "Could not open the file C:\MyProgram\datafile.txt."
    End  ' abort the program
End If

' Get the various times and dates associated with the file:
retval = GetFileTime(hfile, ctime, atime, wtime)
' Load jantime with the date January 5, 1999 at midnight:
jantime.wMonth = 1: jantime.wDay = 5: jantime.wYear = 1999
jantime.wHour = 0: jantime.wMinute = 0: jantime.wSecond = 0
' Convert jantime into FILETIME format so it can be compared with ctime:
retval = SystemTimeToFileTime(jantime, janfiletime)
' Compare the two times and display the relation:
comptimes = CompareFileTime(ctime, janfiletime)
If comptimes = -1 Then Debug.Print "File was created before midnight, January 5, 1999."
If comptimes = 0 Then Debug.Print "File was created at midnight, January 5, 1999."
If comptimes = 1 Then Debug.Print "File was created after midnight, January 5, 1999."
' Close the file
retval = CloseHandle(hfile)

Category: Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CompareString Function

Declare Function CompareString Lib "kernel32.dll" Alias "CompareStringA" (ByVal Locale As Long, ByVal dwCmpFlags As Long, ByVal lpString1 As String, ByVal cchCount1 As Long, ByVal lpString2 As String, ByVal cchCount2 As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CompareString compares two strings and determines which one would come first in an alphabetic sort. Although this function can use a number of different comparisons, by default it conducts a case-sensitive word sort. In a word sort, all symbols except hyphens and apostrophes come before the letter "a" (hyphens and apostrophes are treated differently). The function compares strings by first comparing their first characters, then their second characters, etc. until an unequal pair of characters is found.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns one of the following flags specifying the result of the comparison:

CSTR_LESS_THAN
The first string is less than the second string (i.e., the first string comes before the second string in alphabetical order).

CSTR_EQUAL
The first string is equal to (but not necessarily identical to) the second string.

CSTR_GREATER_THAN
The first string is greater than the second string.

Visual Basic-Specific Issues

None.

Parameters
Locale

The `locale` identifier of the locale to use to compare the strings. This could also be one of the following flags:

- `LOCALE_SYSTEM_DEFAULT`
  - The system's default locale.
- `LOCALE_USER_DEFAULT`
  - The user's default locale.

`dwCmpFlags`

A combination of the following flags specifying options for the comparison. To use the default comparison, set this parameter to 0.

- `NORM_IGNORECASE`
  - Conduct a case-insensitive search.
- `NORM_IGNOREKANATYPE`
  - For Japanese characters, do not differentiate between Hiragana and Katakana characters.
- `NORM_IGNORENONSPACE`
  - Ignore nonspacing characters.
- `NORM_IGNORESYMBOLS`
  - Ignore symbols.
- `NORM_IGNOREWIDTH`
  - Do not differentiate between equivalent single-byte and double-byte characters.
- `SORT_STRINGSORT`
  - Use a string sort instead of a word sort. In a string sort, all symbols, including hyphens and apostrophes, come before the letter "a".

`lpString1`

The first string to compare.

`cchCount1`

The length of `lpString1`. If this is -1, `lpString1` must end in a terminating null character.

`lpString2`

The second string to compare.

`cchCount2`

The length of `lpString2`. If this is -1, `lpString2` must end in a terminating null character.

Constant Definitions

- `Const CSTR_LESS_THAN = 1`
- `Const CSTR_EQUAL = 2`
- `Const CSTR_GREATER_THAN = 3`
- `Const LOCALE_SYSTEM_DEFAULT = &H400`
- `Const LOCALE_USER_DEFAULT = &H800`
- `Const NORM_IGNORECASE = &H1`
- `Const NORM_IGNOREKANATYPE = &H10000`
- `Const NORM_IGNORENONSPACE = &H2`
- `Const NORM_IGNORESYMBOLS = &H4`
- `Const NORM_IGNOREWIDTH = &H20000`
- `Const SORT_STRINGSORT = &H1000`

Example

```
' This code is licensed according to the terms and conditions listed here.

' Use a case-sensitive, string sort comparison method to alphabetically
```
sort nine words. The sorting method simply compares each possible 
pair of words; if a pair is out of alphabetical order, they are switched. 
(Compare these results to those from using lstrcmp for the comparison.)

Dim words(1 To 9) As String  ' the words to sort
Dim tempstr As String  ' buffer used to swap strings
Dim oc As Integer, ic As Integer  ' counter variables
Dim compval As Long  ' result of comparison
Dim threadlocale As Long  ' locale ID of this thread

' Get the locale of this thread (i.e., of this program).
threadlocale = GetThreadLocale()

' Load the nine strings into the array.
words(1) = "can't"
words(2) = "cant"
words(3) = "cannot"
words(4) = "pants"
words(5) = "co-op"
words(6) = "coop"
words(7) = "Denver"
words(8) = "denver"
words(9) = "denveR"

' Sort the strings, swapping any pairs which are out of order.
For oc = 1 To 8  ' first string of the pair
    For ic = oc + 1 To 9  ' second string of the pair
        ' Compare the two strings.
        compval = CompareString(threadlocale, SORT_STRINGSORT, words(oc), Len(words(oc)), words(ic), Len(words(ic)))
        ' If words(oc) is greater, swap them.
        If compval = CSTR_GREATER_THAN Then
            tempstr = words(oc)
            words(oc) = words(ic)
            words(ic) = tempstr
        End If
    Next ic
Next oc

' Display the list of sorted words.
For oc = 1 To 9
    Debug.Print words(oc)
Next oc

See Also
lstrcmp, lstrcmpi

Category
Strings
connect Function

Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

connect establishes a connection between a socket and a network host. The Winsock implementation automatically handles the details of the connection, which vary depending on the base protocols being used (e.g., TCP/IP vs. UDP/IP). Once the connection is established, data can be sent to and from the network host.

Return Value

If successful, the function returns zero. If an error occurred, the function returns SOCKET_ERROR (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- `s` A descriptor of the socket to use to connect. The descriptor is obtained by a previous call to socket.
- `name` Information describing the network host to connect to.
- `namelen` The length of the structure passed as `name`.

Constant Definitions
Const SOCKET_ERROR = -1

Example

Download the main page of this web site (http://www.vbapi.com). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer
Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long
Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
End Type
Windows API Guide: connect Function

```
    sin_addr As Long
    sin_zero As String * 8
End Type

Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long

Declare Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long

Public Const FIONBIO = &H8004667E

Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

Public Const SOCKET_ERROR = -1

' Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA ' info about Winsock
    Dim sock As Long          ' the socket descriptor
    Dim pHostinfo As Long     ' pointer to info about the host computer
    Dim hostinfo As hostent   ' info about the host computer
    Dim pIPAddress As Long    ' pointer to host's IP address
    Dim ipAddress As Long     ' host's IP address
    Dim sockinfo As sockaddr  ' settings for the socket
    Dim buffer As String      ' buffer for sending and receiving data
    Dim reply As String       ' accumulates server's reply
    Dim retval As Long        ' generic return value

    ' Begin a Winsock session.
    retval = WSAStartup(MAKEWORD(2, 2), wsockinfo)
    If retval <> 0 Then
        Debug.Print "Unable to initialize Winsock! --"; retval
        Exit Sub
    End If

    ' Get information about the server to connect to.
    pHostinfo = gethostbyname("www.vbapi.com")
    If pHostinfo = 0 Then
```
Debug.Print "Unable to resolve host!"
GoTo Cleanup
End If
' Copy information about the server into the structure.
CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
If hostinfo.h_addrtype <> AF_INET Then
    Debug.Print "Couldn't get IP address of www.vbapi.com!"
    GoTo Cleanup
End If
' Get the server's IP address out of the structure.
CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4

' Create a socket.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If

' Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
    ' Use Internet Protocol (IP)
    .sin_family = AF_INET
    ' Connect to port 80.
    .sin_port = htons(80)
    ' Connect to this IP address.
    .sin_addr = ipAddress
    ' Padding characters.
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If

' Send an HTTP/GET request for the /index.html file.
buffer = "GET / HTTP/1.1" & vbCrLf & _
    "Host: www.vbapi.com" & vbCrLf & _
    "User-Agent: HTTP-Test-Program" & vbCrLf & vbCrLf
retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request.  Waiting for reply..."

' Make the socket non-blocking, so calls to recv don't halt the program
waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)

' Read the response from the other system. A more sophisticated program
' would watch to see if the connection ever times out (i.e., if the connection is
' lost). For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If
' Process background events so the program doesn't appear to freeze.
    DoEvents
Loop Until retval = 0

' Print the response from the server.
    Debug.Print "Document Retrieved:"
    Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()
End Sub

Category

Winsock

Back to the Function list.
Back to the Reference section.

| Last Modified: January 21, 2001 |
| This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000 |
| Go back to the Windows API Guide home page. |
| E-mail: vbapi@vbapi.com Send Encrypted E-Mail |
| This page is at http://www.vbapi.com/ref/c/connect.html |
CopyFile Function

Declare Function CopyFile Lib "kernel32.dll" Alias "CopyFileA" (ByVal lpExistingFileName As String, ByVal lpNewFileName As String, ByVal bFailIfExists As Long) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

**CopyFile** copies a file from one location to another, just like copying a file in Windows Explorer or in some other way. Depending on the value for **bFailIfExists**, it will either overwrite the target file if it already exists, or will fail. The function returns 1 if successful, or 0 if an error occurred.

**lpExistingFileName**
The source file; i.e., the file to copy from.

**lpNewFileName**
The target file; i.e., the new file to create that will be the copy.

**bFailIfExists**
If 0, the function will overwrite **lpNewFileName** if it already exists. If non-zero, the function will instead fail.

**Example:**

' Copy the file C:\MyStuff\temp.txt to C:\Junk\buffer.txt.
' Do not overwrite C:\Junk\buffer.txt if it already exists.
Dim retval As Long ' return value

' copy the file
retval = CopyFile("C:\MyStuff\temp.txt", "C:\Junk\buffer.txt", 1)
If retval = 0 Then ' failure
    Debug.Print "Copy failed -- C:\Junk\buffer.txt already exists.
Else ' success
    Debug.Print "Copy succeeded."
End If

**See Also:** MoveFile
CopyMemory Function

Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

CopyMemory, as far as the Windows API is concerned, is perfectly identical to the MoveMemory function; in fact, they actually are the same function! CopyMemory moves the contents of a portion of memory from one location to another. The two locations are identified by pointers to the memory addresses. After the copy, the original contents in the source are set to zeros.

Return Value

CopyMemory does not return a value.

Visual Basic-Specific Issues

A pointer to any variable can be automatically generated merely be passing that variable as either Destination or Source. However, if either a String or a Long holding the desired memory address is passed, the ByVal keyword must precede it.

Parameters

Destination
A pointer to the memory address to use as the target, which receives the transferred data.

Source
A pointer to the memory address to use as the source, which initially holds the data to be transferred.

Length
The number of bytes of data to copy from the source memory location to the target memory location.
Example

' This code is licensed according to the terms and conditions listed here.

' Transfer the contents of one byte array to another. After the "copy",
' the contents of the source array are set to 0.
Dim source(0 To 9) As Byte ' source array of 10 bytes
Dim target(0 To 9) As Byte ' similarly sized target array
Dim c As Integer ' counter variable

' Fill the source array with some information.
For c = 0 To 9 ' loop through each element
    source(c) = c ' set each element's value to its index
Next c

' Transfer the data from the target array to the source array. Note how pointers
' are implied merely by passing the arrays as usual.
CopyMemory target(0), source(0), 10 ' copy all 10 bytes

' Verify that the contents were transferred.
For c = 0 To 9
    Debug.Print target(c); ' this will now contain the information
Next c

See Also

MoveMemory

Category

Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: July 28, 1999
This page is copyright © 1999 Paul Kulimiewicz. Copyright Information
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/copymemory.html
CopyRect Function

Declare Function CopyRect Lib "user32.dll" (lpDestRect As RECT, lpSourceRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

CopyRect sets one rectangle equal to another. This is done by duplicating all of the source rectangle's member values to the corresponding ones in the target rectangle. This is faster than setting all four values manually in your code. The function returns 0 if an error occurred, or 1 if successful.

lpDestRect
The target rectangle to set.

lpSourceRect
The source rectangle.

Example:

' Set the source and target rectangles equal to the rectangle ' of the window by copying the source to the target
Dim source As RECT, target As RECT ' source & target rectangles
Dim retval As Long ' return value

' Get the rectangle of Form1 into source
retval = GetWindowRect(Form1.hWnd, source)
' Copy source into target
retval = CopyRect(target, source)

See Also: EqualRect
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CoTaskMemFree Function

Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal pv As Long)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 2.0 or later.

Description & Usage

**CoTaskMemFree** frees some types of blocks of allocated memory. This function will free the memory associated with a pointer to an ITEMIDLIST structure (a.k.a. a PIDL) which was created by a Windows shell API function. This function must be used on such blocks of memory to free system resources.

Return Value

**CoTaskMemFree** does not return a value.

Visual Basic-Specific Issues

None.

Parameters

`pv`  
A pointer to the memory block to free.

Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/c/cotaskmemfree.html).

' Open the Browse for Folder dialog box and display both the display name and the actual name of the folder (if it is not a virtual folder). Any folder under My Computer may be selected.
Dim bi As BROWSEINFO  ' structure passed to the function
Dim pidl As Long  ' PIDL to the user's selection
Dim physpath As String  ' string used to temporarily hold the physical path
Dim retval As Long  ' return value

' Initialize the structure to be passed to the function.
bi.hwndOwner = Form1.hWnd  ' window Form1 is the owner of the dialog box
' Specify the My Computer virtual folder as the root
tval = SHGetSpecialFolderLocation(Form1.hWnd, CSIDL_DRIVES, bi.pidlRoot)
' Make room in the buffer to get the [virtual] folder's display name
bi.pszDisplayName = Space(260)
bi.lpszTitle = "Please choose a folder."  ' Message displayed to the user
bi.ulFlags = 0  ' no flags are needed here
bi.lpfn = 0  ' no callback function is being used
bi.lParam = 0  ' not needed
bi.iImage = 0  ' this will be set by the function

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
   ' Remove the empty space from the display name variable.
   bi.pszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
   Debug.Print "The user selected: "; bi.pszDisplayName
   ' If the folder is not a virtual folder, display its physical location.
   physpath = Space(260)  ' make room in the buffer
   retval = SHGetPathFromIDList(pidl, physpath)
   If retval = 0 Then
      Debug.Print "Physical Location: (virtual folder)"
   Else
      ' Remove the empty space and display the result.
      physpath = Left(physpath, InStr(physpath, vbNullChar) - 1)
      Debug.Print "Physical Location: "; physpath
   End If
' Free the pidl returned by the function.
CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to
' identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

Category

OLE
CreateCursor Function

Declare Function CreateCursor Lib "user32.dll" (ByVal hInstance As Long, ByVal nXhotspot As Long, ByVal nYhotspot As Long, ByVal nWidth As Long, ByVal nHeight As Long, lpANDbitPlane As Any, lpXORbitPlane As Any) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateCursor creates a new cursor. Its image is formed by using an AND mask and a XOR mask provided by the function, which are used to place the cursor's image wherever it appears. The AND and XOR masks can be provided using any numeric array, such as a Byte array (as the example below uses). The function also specifies the cursor's hotspot, the exact pixel which is considered to be the location of the cursor (such as the exact tip of the arrow cursor). Note that it is usually a better idea to put cursors into some sort of resource or separate file and load them instead of hard-wiring the cursors into the application code via this function. (This is because this function relies on creating a cursor compatible with the display device, instead of generating one from a file or resource which is.) Of course, the cursor size must be one supported by the system; use GetSystemMetrics to check. The cursor created by this function must later be destroyed by using DestroyCursor. The function returns 0 if an error occured, or a handle to the newly created cursor if successful.

hInstance
The instance handle of the application which is calling the function.

nXhotspot
The x-coordinate of the cursor's hotspot, relative to the cursor's upper-left corner.

nYhotspot
The y-coordinate of the cursor's hotspot, relative to the cursor's upper-left corner.

nWidth
The width in pixels of the cursor.

nHeight
The height in pixels of the cursor.

lpANDbitPlane
An array holding the AND mask for the cursor's image.

lpXORbitPlane
An array holding the XOR mask for the cursor's image.

Example:

' Create a 32x32 color cursor shaped somewhat like a yin-yang symbol.
' (The bit masks come from Microsoft's documentation on the API cursors function, just to
' give them their due credit.) Note how the masks are loaded into the arrays. The new
' cursor is then set to be the cursor for 10 seconds.
Dim hnewcursor As Long ' newly created cursor
Dim holdcursor As Long ' receives handle of default cursor
Dim andbuffer As String, xorbuffer As String  ' buffers for masks
Dim andbits(0 To 127) As Byte  ' stores the AND mask
Dim xorbits(0 To 127) As Byte  ' stores the XOR mask
Dim c As Integer, retval As Long  ' counter and return value

' Unfortunately, VB does not provide a nice way to load lots of information into an array.
' To load the AND and XOR masks, we put the raw hex values into the string buffers and use a loop to convert the hex values into numeric values and load them into the elements of the array. Yes, it's ugly, but there's no better way. Note the use of the line-continuation character here. Each sequence of eight hex characters represents one line in the 32x32 cursor.
andbuffer = "FFFC3FFF" & "FFC01FFF" & "FF003FFF" & "FE00FFFF" & "F701FFFF" & "F003FFFF" & "E007FFFF" & "80007FFF" & "800007FF" & "C0007FFF" & "C0000FFF" & "E0000FFF" & "F0001FFF" & "F8003FFF" & "F8000000" & "00000000" & "0003C000" & "003F0000" & "00FE0000" & "00F80000" & "07F80000" & "07FF80000" & "3FE00000" & "3FF80000" & "7FF80000" & "7FFC0000" & "7FFE0000" & "3FFE0000" & "1F83E000" & "01FF8000" & "00 FF0000" & "003C0000" & "00000000"
 xorbuffer = "00000000" & "0003C000" & "003F0000" & "00FE0000" & "00F80000" & "07F80000" & "07FF80000" & "3FE00000" & "3FF80000" & "7FF80000" & "7FFC0000" & "7FFE0000" & "3FFE0000" & "1F83E000" & "01FF8000" & "00 FF0000" & "003C0000" & "00000000"

' Now load these hex values into the proper arrays.
For c = 0 To 127
    andbits(c) = "&H" & Mid(andbuffer, 2 * c + 1)
    xorbits(c) = "&H" & Mid(xorbuffer, 2 * c + 1)
Next c

' Finally, create this cursor! The hotspot is at (19,2) on the cursor.
hnewcursor = CreateCursor(App.hInstance, 19, 2, 32, 32, andbits(0), xorbits(0))
' Set the new cursor as the current cursor for 10 seconds and then switch back.
holdcursor = SetCursor(hnewcursor)  ' change cursor
Sleep 10000  ' wait for 10 seconds
retval = SetCursor(holdcursor)  ' change cursor back
' Destroy the new cursor.
retval = DestroyCursor(hnewcursor)

See Also: DestroyCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreateDC Function

Declare Function CreateDC Lib "gdi32.dll" Alias "CreateDCA" (ByVal lpszDriver As String, ByVal lpszDevice As String, ByVal lpszOutput As Long, lpInitData As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

CreateDC creates a device context to a given object. The object is identified by its name. When you are finished using the device context in your program, use the DeleteDC function to destroy it. Do not use ReleaseDC with device contexts created by this function!

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a handle to the newly created device context.

Visual Basic-Specific Issues

When passing 0 for lpInitData, the expression ByVal CLng(0) must be used.

Parameters

lpszDriver
This string should normally be empty, except for using "DISPLAY" to reference the display driver. Windows NT, 2000: This could also be "WINSPOOL" to reference the print spooler.

lpszDevice
The name of the device to create a device context for.

lpszOutput
Reserved -- set to 0.

lpInitData
If desired, a DEVMODE structure holding device initialization information for the device. To use the default
Example:

' Print out a page with an ellipse drawn with a thickened black pen on it. The page is printed on the computer's default printer.

' The following are special declarations needed to allow string manipulation functions to use pointers to strings.
Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

' Variable declarations
Dim hPrintDC As Long  ' handle to the printer's device context
Dim di As DOCINFO  ' information about the document to print
Dim hPen As Long  ' handle to the pen to draw the ellipse with
Dim hOldPen As Long  ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long  ' 3076-byte buffer
Dim pi2 As PRINTER_INFO_2  ' receives info about the default printer
Dim printret As Long  ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long  ' receives space requires for EnumPrinters
Dim retval As Long  ' return value

' Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.
retval = EnumPrinters(PRINTER_ENUM_DEFAULT, "", 2, buffer(0), 3076, spaceneeded, printret)
If retval = 0 Then
    Debug.Print "No default printer is configured."
    End  ' abort the program
End If

' Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.
pi2.pPrinterName = Space(lstrlen(buffer(1)))
retval = lstrcpy(pi2.pPrinterName, buffer(1))
pi2.pDriverName = Space(lstrlen(buffer(4)))
retval = lstrcpy(pi2.pDriverName, buffer(4))

' Create a device context to the printer, using its default initialization.
hPrintDC = CreateDC("", pi2.pPrinterName, 0, ByVal CLng(0))
' Create a solid black brush with a thickness of 5.
hPen = CreatePen(PS_SOLID, 5, RGB(0, 0, 0))

' Load information about the document to print into the structure.
di.cbSize = Len(di)  ' size of structure
di.lpszDocName = "Printer API Demonstration"  ' name of document
di.lpszOutput = 0  ' do not print to a file
di.lpszDatatype = ""  ' data type of file doesn't apply
di.fwType = 0 ' no additional information

' Begin the print job.
retval = StartDoc(hPrintDC, di)
' Begin the first and only page to print.
retval = StartPage(hPrintDC)
' Select the pen for use with the printer.
   hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

DeleteDC

Category

Devices

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

Last Modified: November 6, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/createdc.html
CreateDirectory Function

Declare Function CreateDirectory Lib "kernel32.dll" Alias "CreateDirectoryA" (ByVal lpPathName As String, lpSecurityAttributes As SECURITY_ATTRIBUTES) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreateDirectory creates a new directory on a disk. It also allows you to specify the security attributes of the newly created directory, if the operating system supports it.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- lpPathName
  The name of the new directory to create.
- lpSecurityAttributes
**Windows NT, 2000:** The security attributes to assign to the newly created directory. **Windows 95, 98, CE:** This parameter is ignored.

### Example

`' This code is licensed according to the terms and conditions listed [here](http://www.vbapi.com/ref/c/createdirectory.html).

' Create the new directory C:\Dummy\NewDir and
give it default security attributes.
Dim secattr As SECURITY_ATTRIBUTES ' security attributes structure
Dim retval As Long ' return value

' Set the desired security attributes
secattr.nLength = Len(secattr) ' size of the structure
secattr.lpSecurityDescriptor = 0 ' default (normal) level of security
secattr.bInheritHandle = 1 ' this is the default setting

' Create the directory.
retval = CreateDirectory("C:\Dummy\NewDir", secattr)

### See Also

CreateDirectoryEx, RemoveDirectory

### Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** March 19, 2000
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/c/createdirectory.html) Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com [Send Encrypted E-Mail](http://www.vbapi.com/ref/c/createdirectory.html)
This page is at [http://www.vbapi.com/ref/c/createdirectory.html](http://www.vbapi.com/ref/c/createdirectory.html)
CreateDirectoryEx Function

Declare Function CreateDirectoryEx Lib "kernel32.dll" Alias "CreateDirectoryExA" (ByVal lpTemplateDirectory As String, ByVal lpNewDirectory As String, lpSecurityAttributes As SECURITY_ATTRIBUTES) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Not Supported.

Description & Usage

CreateDirectoryEx creates a new directory on a disk. It also allows you to specify the security attributes of the newly created directory, if the operating system supports it. The newly created directory will inherit most of its attributes (except security) from a template directory specified by the function. For example, the new directory will have the same file attributes as the template directory.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpTemplateDirectory
The name of the directory to use as an attribute template for creating the new directory.

lpNewDirectory
The name of the new directory to create.
Windows API Guide: CreateDirectoryEx Function

**lpSecurityAttributes**

**Windows NT, 2000:** The security attributes to assign to the newly created directory. **Windows 95, 98, CE:** This parameter is ignored.

### Example

' This code is licensed according to the terms and conditions listed [here](#).

' Create the new directory C:\Dummy\NewDir and
give it default security attributes. It will inherit its properties from the
directory C:\Recycled (the "Recycle Bin") -- although this won't be another
'recycle bin, it will have the Hidden and System attributes.
Dim secattr As SECURITY_ATTRIBUTES ' security attributes structure
Dim retval As Long ' return value

' Set the desired security attributes
secattr.nLength = Len(secattr) ' size of the structure
secattr.lpSecurityDescriptor = 0 ' default (normal) level of security
secattr.bInheritHandle = 1 ' this is the default setting

' Create the directory, using C:\Recycled as the template.
retval = CreateDirectoryEx("C:\Recycled", "C:\Dummy\NewDir", secattr)

### See Also

CreateDirectory, RemoveDirectory

### Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** March 19, 2000
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/c/createdirectory.html](http://www.vbapi.com/ref/c/createdirectory.html)
CreateEllipticRgn Function

Declare Function CreateEllipticRgn Lib "gdi32.dll" (ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

**CreateEllipticRgn** creates an elliptically-shaped region. The ellipse which forms the region is specified by the bounding rectangle defined by the coordinates passed to the function. The bounding rectangle is the smallest possible rectangle which can fit around the ellipse. The function returns a handle to the newly created region if successful, or 0 if an error occurred.

- **X1**: The x-coordinate of the upper-left corner of the ellipse's bounding rectangle.
- **Y1**: The y-coordinate of the upper-left corner of the ellipse's bounding rectangle.
- **X2**: The x-coordinate of the lower-right corner of the ellipse's bounding rectangle.
- **Y2**: The y-coordinate of the lower-right corner of the ellipse's bounding rectangle.

**Example:**

```
' Invert the pixels within an elliptical region on window Form1.  The elliptical region has a bounding rectangle of (20,30)-(150,110).
Dim hrgn As Long  ' handle to the region to invert
Dim retval As Long  ' return value

' Create the elliptical region to invert and get a handle to it.
hrgn = CreateEllipticRgn(20,30,150,110)  ' bounding rectangle (20,30)-(150,110)
' Invert that region in window Form1.
retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = DeleteObject(hrgn)
```

**See Also:** CreateEllipticRgnIndirect

**Category:** Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreateEllipticRgnIndirect Function

Declare Function CreateEllipticRgnIndirect Lib "gdi32.dll" (lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateEllipticRgn creates an elliptically-shaped region. The ellipse which forms the region is specified by the bounding rectangle defined by the rectangle structure passed to the function. The bounding rectangle is the smallest possible rectangle which can fit around the ellipse. The function returns a handle to the newly created region if successful, or 0 if an error occurred.

lpRect
The bounding rectangle of the ellipse which forms the region.

Example:

' Invert the pixels within an elliptical region within window Form1. The elliptical region has a bounding rectangle of (20,30)-(150,110)
Dim hrgn As Long ' handle to the elliptical region
Dim bounding As RECT ' bounding rectangle
Dim retval As Long ' return value

' Load the coordinates of the bounding rectangle into the structure.
retval = SetRect (bounding, 20, 30, 150, 110) ' bounding = (20,30)-(150,110)
' Create the elliptical region from this bounding rectangle.
hrgn = CreateEllipticRgnIndirect (bounding)
' Invert the pixels on Form1 within the region.
retval = InvertRgn (Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = DeleteObject (hrgn)

See Also: CreateEllipticRgn
Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreateFile Function

Declare Function CreateFile Lib "kernel32.dll" Alias "CreateFileA" (ByVal lpFileName As String, ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long, ByVal lpSecurityAttributes As Any, ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As Long, ByVal hTemplateFile As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreateFile creates or opens a console, communications resource, directory (can only open), disk devices (Windows NT, 2000 only), files, mailslots, and pipes. Obviously, files are the most common thing opened by the function. The object opened can then be read from or written to, as the access level allows. After your program is finished using the handle generated by the function, it must call CloseHandle to close the open object.

Return Value

If an error occurred, the function returns -1 (use GetLastError to get the error code). If successful, the function returns a handle to the opened or created file or other object.

Visual Basic-Specific Issues

When passing 0 as the lpSecurityAttributes parameter, the expression ByVal CLng(0) must be used to pass the zero correctly. See the example for a demonstration.

Parameters

lpFileName
The name of the file or other allowed object to create or open.

dwDesiredAccess
A combination of the following flags (if any) specifying the amounts of read/write access to the file or other object:

- GENERIC_READ
  Allow the program to read data from the file or other object.
GENERIC_WRITE
Allow the program to write data to the file or other object.

dwShareMode
A combination of the following flags (if any) specifying the amounts of read/write access to grant other programs attempting to access the file or other object while your program still has it open:
FILE_SHARE_READ
Allow other programs to read data from the file or other object.
FILE_SHARE_WRITE
Allow other programs to write data to the file or other object.

lpSecurityAttributes
Windows NT, 2000: A SECURITY_ATTRIBUTES structure specifying the security attributes to give the created or opened file or other object. Windows 95, 98, CE: This parameter must be 0.

dwCreationDisposition
Exactly one of the following flags specifying how and when to create or open the file or other object depending if it already does or does not exist:
CREATE_ALWAYS
Create a new file or other object. Overwrite the file or other object (i.e., delete the old one first) if it already exists.
CREATE_NEW
Create a new file or other object. The function fails if the file or other object already exists.
OPEN_ALWAYS
Open an existing file or other object. If the file or other object does not exist, it will be created.
OPEN_EXISTING
Open an existing file or other object. The function fails if the file or other object does not exist.
TRUNCATE_EXISTING
Open an existing file or other object and delete its contents. The function fails if the file or other object does not exist.

dwFlagsAndAttributes
The combination of the following flags specifying both the file attributes of a newly created file and other options for creating or opening the file. One flag specifying the file attributes must be included.
FILE_ATTRIBUTE_ARCHIVE
An archive file (which most files are).
FILE_ATTRIBUTE_HIDDEN
A hidden file, not normally visible to the user.
FILE_ATTRIBUTE_NORMAL
An attribute-less file (cannot be combined with other attributes).
FILE_ATTRIBUTE_READONLY
A read-only file.
FILE_ATTRIBUTE_SYSTEM
A system file, used exclusively by the operating system.
FILE_FLAG_DELETE_ON_CLOSE
Delete the file once it is closed.
FILE_FLAG_NO_BUFFERING
Do not use any buffers or caches. If used, the following things must be done: access to the file must begin at whole number multiples of the disk's sector size; the amounts of data accessed must be a whole number multiple of the disk's sector size; and buffer addresses for I/O operations must be aligned on whole number multiples of the disk's sector size.
FILE_FLAG_OVERLAPPED
Allow asynchronous I/O; i.e., allow the file to be read from and written to simultaneously. If used, functions that read and write to the file must specify the OVERLAPPED structure identifying the file pointer. (Windows 95, 98, CE: Overlapped files are not supported, although other overlapped objects are.)

FILE_FLAG_POSIX_SEMANTICS
Allow file names to be case-sensitive.

FILE_FLAG_RANDOM_ACCESS
Optimize the file cache for random access (skipping around to various parts of the file).

FILE_FLAG_SEQUENTIAL_SCAN
Optimize the file cache for sequential access (starting at the beginning and continuing to the end of the file).

FILE_FLAG_WRITE_THROUGH
Bypass any disk cache and instead read and write directly to the file.

hTemplateFile
A handle to an open file to copy the attributes of, or 0 to not copy the attributes of any open file.

Constant Definitions

Const GENERIC_READ = &H80000000
Const GENERIC_WRITE = &H40000000
Const FILE_SHARE_READ = &H1
Const FILE_SHARE_WRITE = &H2
Const CREATE_ALWAYS = 2
Const CREATE_NEW = 1
Const OPEN_ALWAYS = 4
Const OPEN_EXISTING = 3
Const TRUNCATE_EXISTING = 5
Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H80
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4
Const FILE_FLAG_DELETE_ON_CLOSE = &H4000000
Const FILE_FLAG_NO_BUFFERING = &H20000000
Const FILE_FLAG_OVERLAPPED = &H40000000
Const FILE_FLAG_POSIX_SEMANTICS = &H1000000
Const FILE_FLAG_RANDOM_ACCESS = &H10000000
Const FILE_FLAG_SEQUENTIAL_SCAN = &H8000000
Const FILE_FLAG_WRITE_THROUGH = &H80000000

Example

' This code is licensed according to the terms and conditions listed here.

' Display the date on which the file C:\MyApp\test.txt was created. Note how the time zone conversion is necessary.
Dim hFile As Long ' handle to the opened file
Dim ctime As FILETIME ' receives time of creation
Dim atime As FILETIME ' receives time of last access
Dim mtime As FILETIME ' receives time of last modification
Dim thetime As SYSTEMTIME ' used to manipulate the time
Dim retval As Long ' return value

' First, open the file C:\MyApp\test.txt for read-level access. Note the ' expression necessary to pass 0 as lpSecurityAttributes.
hFile = CreateFile("C:\MyApp\test.txt", GENERIC_READ, FILE_SHARE_READ, ByVal CLng(0),
If hFile = -1 Then
    Debug.Print "Could not open the file successfully -- aborting."
    End ' terminate the program
End If

' Next, get the creation, last-access, and last-modification times.
retval = GetFileTime(hFile, ctime, atime, mtime)
' Convert the creation time to the local time zone.
retval = FileTimeToLocalFileTime(ctime, ctime)
' Convert the FILETIME format to the SYSTEMTIME format.
retval = FileTimeToSystemTime(ctime, thetime)

' Display the date of creation of the file to the user.
Debug.Print "The file was created on "; thetime.wMonth; "-"; thetime.wDay; "-"; thetime.wYear

' Close the file to free up resources.
retval = CloseHandle(hFile)

Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 30, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/createfile.html

CreateFont Function

Declare Function CreateFont Lib "gdi32.dll" Alias "CreateFontA" (ByVal nHeight As Long, ByVal nWidth As Long, ByVal nEscapement As Long, ByVal nOrientation As Long, ByVal fnWeight As Long, ByVal fdwItalic As Long, ByVal fdwUnderline As Long, ByVal fdwStrikeOut As Long, ByVal fdwCharSet As Long, ByVal fdwOutputPrecision As Long, ByVal fdwClipPrecision As Long, ByVal fdwQuality As Long, ByVal fdwPitchAndFamily As Long, ByVal lpszFace As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

CreateFont creates a font object, which allows a font with given attributes to be used to draw text on a device. The font created by the function is the one which most closely matches the attributes of the logical font information passed via the numerous parameters. After your program is finished using the font, it must be deleted by using DeleteObject.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a handle to the font just created.

Visual Basic-Specific Issues

None.

Parameters

nHeight

The height of the font's character cell, in logical units (also known as the em height). If positive, the font mapper converts this value directly into device units and matches it with the cell height of the possible fonts. If 0, the font mapper uses a default character height. If negative, the font mapper converts the absolute value into device units and matches it with the character height of the possible fonts.
The average width of the font's characters. If 0, the font mapper tries to determine the best value.

The angle between the font's baseline and escapement vectors, in units of $\frac{1}{10}$ degrees. **Windows 95, 98:** This must be equal to \lfOrientation.  

The angle between the font's baseline and the device's x-axis, in units of $\frac{1}{10}$ degrees. **Windows 95, 98:** This must be equal to \lfEscapement.

One of the following flags specifying the boldness (weight) of the font:
- FW_DONTCARE: Default weight.
- FW_THIN: Thin weight.
- FW_EXTRALIGHT: Extra-light weight.
- FW_ULTRALIGHT: Same as FW_EXTRALIGHT.
- FW_LIGHT: Light weight.
- FW_NORMAL: Normal weight.
- FW_REGULAR: Same as FW_NORMAL.
- FW_MEDIUM: Medium weight.
- FW_SEMIBOLD: Semi-bold weight.
- FW_DEMIBOLD: Same as FW_SEMIBOLD.
- FW_BOLD: Bold weight.
- FW_EXTRABOLD: Extra-bold weight.
- FW_ULTRABOLD: Same as FW_EXTRABOLD.
- FW_HEAVY: Heavy weight.
- FW_BLACK: Same as FW_HEAVY.

A non-zero value if the font is italicized, 0 if not.

A non-zero value if the font is underlined, 0 if not.

A non-zero value if the font is striked out, 0 if not.

Exactly one of the following flags specifying the character set of the font:
- ANSI_CHARSET: ANSI character set.
- ARABIC_CHARSET:
Windows API Guide: CreateFont Function


BALTIC_CHARSET

Windows 95, 98: Baltic character set.

CHINESEBIG5_CHARSET

Chinese Big 5 character set.

DEFAULT_CHARSET

Default character set.

EASTEUROPE_CHARSET

Windows 95, 98: Eastern European character set.

GB2312_CHARSET

GB2312 character set.

GREEK_CHARSET

Windows 95, 98: Greek character set.

HANGEUL_CHARSET

HANDEUL character set.

HEBREW_CHARSET


JOHAB_CHARSET

Windows 95, 98: Johab character set.

MAC_CHARSET

Windows 95, 98: Mac character set.

OEM_CHARSET

Original equipment manufacturer (OEM) character set.

RUSSIAN_CHARSET

Windows 95, 98: Russian character set.

SHIFTJIS_CHARSET

ShiftJis character set.

SYMBOL_CHARSET

Symbol character set.

THAI_CHARSET

Windows NT, 2000: Thai character set.

TURKISH_CHARSET

Windows 95, 98: Turkish character set.

fdwOutPrecision

Exactly one of the following flags specifying the desired precision (closeness of the match) between the logical font ideally described by the structure and the actual logical font. This value is used by the font mapper to produce the logical font.

OUT_DEFAULT_PRECIS

The default font mapping behavior.

OUT_DEVICE_PRECIS

Choose a device font if there are multiple fonts in the system with the same name.

OUT_OUTLINE_PRECIS

Windows NT, 2000: Choose a TrueType or other outline-based font.

OUT_RASTER_PRECIS

Choose a raster font if there are multiple fonts in the system with the same name.

OUT_STRING_PRECIS

Raster font (used for enumeration only).

OUT_STROKE_PRECIS

Windows 95, 98: Vector font (used for enumeration only). Windows NT, 2000: TrueType, outline-based, or vector font (used for enumeration only).

OUT_TT_ONLY_PRECIS
Choose only a TrueType font.

OUT_TT_PRECIS
Choose a TrueType font if there are multiple fonts in the system with the same name.

**fdwClipPrecision**
Exactly one of the following flags specifying the clipping precision to use when the font's characters must be clipped:

- CLIP_DEFAULT_PRECIS
  The default clipping behavior.
- CLIP_EMBEDDED
  This flag must be set for an embedded read-only font.
- CLIP_LH_ANGLES
  The direction of any rotations is determined by the coordinate system (or else all rotations are counterclockwise).
- CLIP_STROKE_PRECIS
  Raster, vector, or TrueType font (used for enumeration only).

**fdwQuality**
Exactly one of the following flags specifying the output quality of the logical font as compared to the ideal font:

- ANTIALIASED_QUALITY
  Windows 95, 98, NT 4.0 or later, 2000: The font is always antialiased if possible.
- DEFAULT_QUALITY
  The default quality: the appearance of the font does not matter.
- DRAFT_QUALITY
  The appearance of the font is less important then in PROOF_QUALITY.
- NONANTIALIASED_QUALITY
  Windows 95, 98, NT 4.0 or later, 2000: The font is never antialiased.
- PROOF_QUALITY
  The quality of the appearance of the font is more important than exactly matching the specified font attributes.

**fdwPitchAndFamily**
A bitwise OR combination of exactly one *_PITCH flag specifying the pitch of the font and exactly one FF_* flag specifying the font face family of the font:

- DEFAULT_PITCH
  The default pitch.
- FIXED_PITCH
  Fixed pitch.
- VARIABLE_PITCH
  Variable pitch.
- FF_DECORATIVE
  Showy, decorative font face.
- FF_DONTCARE
  Do not care about the font face.
- FF_MODERN
  Modern font face (monospaced, sans serif font).
- FF_ROMAN
  Roman font face (proportional-width, serif font).
- FF_SCRIPT
  Script font face which imitates script handwriting.
- FF_SWISS
  Swiss font face (proportional-width, sans serif font).

**lpszFace**
The name of the font face to use.

---

**Constant Definitions**

Const FW_DONTCARE = 0
Const FW_THIN = 100
Const FW_EXTRALIGHT = 200
Const FW_ULTRALIGHT = 200
Const FW_LIGHT = 300
Const FW_NORMAL = 400
Const FW_REGULAR = 400
Const FW_MEDIUM = 500
Const FW_SEMIBOLD = 600
Const FW_DEMIBOLD = 600
Const FW_BOLD = 700
Const FW_EXTRABOLD = 800
Const FW_ULTRABOLD = 800
Const FW_HEAVY = 900
Const FW_BLACK = 900
Const ANSI_CHARSET = 0
Const ARABIC_CHARSET = 178
Const BALTIC_CHARSET = 186
Const CHINESEBIG5_CHARSET = 136
Const DEFAULT_CHARSET = 1
Const EASTEUROPE_CHARSET = 238
Const GB2312_CHARSET = 134
Const GREEK_CHARSET = 161
Const HANGEUL_CHARSET = 129
Const HEBREW_CHARSET = 177
Const JOHAB_CHARSET = 130
Const MAC_CHARSET = 77
Const OEM_CHARSET = 255
Const RUSSIAN_CHARSET = 204
Const SHIFTJIS_CHARSET = 128
Const SYMBOL_CHARSET = 2
Const THAI_CHARSET = 222
Const TURKISH_CHARSET = 162
Const OUT_DEFAULT_PRECIS = 0
Const OUT_DEVICE_PRECIS = 5
Const OUT_OUTLINE_PRECIS = 8
Const OUT_RASTER_PRECIS = 6
Const OUT_STRING_PRECIS = 1
Const OUT_STROKE_PRECIS = 3
Const OUT_TT_ONLY_PRECIS = 7
Const OUT_TT_PRECIS = 4
Const CLIP_DEFAULT_PRECIS = 0
Const CLIP_EMBEDDED = 128
Const CLIP_LH_ANGLES = 16
Const CLIP_STROKE_PRECIS = 2
Const ANTIALIASED_QUALITY = 4
Const DEFAULT_QUALITY = 0
Const DRAFT_QUALITY = 1
Const NONANTIALIASED_QUALITY = 3
Const PROOF_QUALITY = 2
Const DEFAULT_PITCH = 0
Const FIXED_PITCH = 1
Const VARIABLE_PITCH = 2
Const FF_DECORATIVE = 80
Const FF_DONTCARE = 0
Const FF_ROMAN = 16
Const FF_SCRIPT = 64
Const FF_SWISS = 32

Example

' This code is licensed according to the terms and conditions listed here.

' Draw the alphabet in the upper-left corner of window Form1 using
' the goofy Symbol font.
Dim hFont As Long ' handle to the Symbol font which is created
Dim hOldFont As Long ' handle to the font previously selected by Form1
Dim retval As Long ' return value

' Create a font object using the Symbol font. Only apply boldface
' formatting to the font -- use defaults for most other settings.
hFont = CreateFont(0, 0, 0, 0, FW_BOLD, 0, 0, 0, SYMBOL_CHARSET, OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH Or FF_DECORATIVE, "Symbol")

' Select that font for use with Form1, saving the previous selection.
hOldFont = SelectObject(Form1.hDC, hFont)
' Draw the alphabet near the upper-left corner.
retval = TextOut(Form1.hDC, 10, 10, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 26)

' Select the previous font back.
retval = SelectObject(Form1.hDC, hOldFont)
' Delete the font we created to free up resources.
retval = DeleteObject(hFont)

See Also

CreateFontIndirect

Category

Fonts & Text

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 13, 1999
CreateFontIndirect Function

Declare Function CreateFontIndirect Lib "gdi32.dll" Alias "CreateFontIndirectA" (lplf As LOGFONT) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreateFontIndirect creates a font object, which allows a font with given attributes to be used to draw text on a device. The font created by the function is the one which most closely matches the attributes of the logical font information passed via the structure. After your program is finished using the font, it must be deleted by using DeleteObject.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a handle to the font just created.

Visual Basic-Specific Issues

None.

Parameters

lplf

Information describing the desired attributes to give to the newly created font.

Example
This code is licensed according to the terms and conditions listed here.

Use the font used by Windows to draw the title text of icons to write the alphabet on window Form1.

```vbp
Dim lf As LOGFONT ' receives information about the font Windows uses
Dim hFont As Long ' handle to the font that is created
Dim hOldFont As Long ' handle to the font which Form1 previously had selected
Dim retval As Long ' return value

' Get the attributes of the logical font used by Windows to draw the title text of icons.
retval = SystemParametersInfo(SPI_GETICONTITLELOGFONT, Len(lf), lf, 0)

' Create a font matching the logical font description.
hFont = CreateFontIndirect(lf)

' Select that font for use in Form1, noting the previous font.
hOldFont = SelectObject(Form1.hDC, hFont)

' Write the alphabet near the upper-left corner of Form1.
retval = TextOut(Form1.hDC, 10, 10, "ABCDEFGHIJKLMNOPQRSTUVWXYZ", 26)

' Restore the previously selected font for use in Form1.
retval = SelectObject(Form1.hDC, hOldFont)

' Delete the created font to free resources.
retval = DeleteObject(hFont)
```

See Also

CreateFont

Category

Fonts & Text

Last Modified: October 11, 1999

This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000

Go back to the alphabetical Function listing.
Go back to the Reference section index.

CreateHatchBrush Function

Declare Function CreateHatchBrush Lib "gdi32.dll" (ByVal nIndex As Long, ByVal crColor As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateHatchBrush creates a hatched brush object. When used to fill an area or shape, this brush produces a pattern of lines (a "hatch pattern") in a single color using an 8x8 unit cell. After the program finishes using the hatched brush, it should use DeleteObject to delete the brush and free system resources. The function returns a handle to the newly created hatched brush if successful, or 0 if an error occurred.

*nIndex*

Exactly one of the following flags specifying which hatch pattern to use to make the brush:
- HS_BDIAGONAL = 3
  - Diagonal lines from the bottom-left to the upper-right.
- HS_CROSS = 4
  - Cross pattern of horizontal and vertical lines.
- HS_DIAGCROSS = 5
  - Cross pattern of perpendicular diagonal lines.
- HS_FDIAGONAL = 2
  - Diagonal lines from the upper-left to the bottom-right.
- HS_HORIZONTAL = 0
  - Horizontal lines.
- HS_VERTICAL = 1
  - Vertical lines.

**crColor**

The RGB value of the color to give the hatched brush. Visual Basic users can use the RGB() function to generate this value.

Example:

' Draw a rectangle with corners (10,20) and (175,100)
' on window Form1. Use a yellow brush with a diagonal cross pattern to fill the rectangle.
Dim hbrush As Long  ' receives handle to the hatched yellow brush
Dim holdbrush As Long  ' receives handle to Form1's default brush
Dim retval As Long  ' return value

hbrush = CreateHatchBrush(HS_DIAGCROSS, RGB(255, 255, 0))  ' create a hatched yellow brush
' Save Form1's default brush so we can restore it after the program is finished
holdbrush = SelectObject(Form1.hDC, hbrush)  ' select the brush
' Draw the rectangle filled using the hatched yellow brush
retval = Rectangle(Form1.hDC, 10, 20, 175, 100)
' Restore Form1's previous brush before destroying the created one
retval = SelectObject(Form1.hDC, holdbrush)  ' select old brush
retval = DeleteObject(hbrush)  ' destroy the hatched yellow brush

See Also: CreateSolidBrush
Category: Brushes

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreatePen Function

Declare Function CreatePen Lib "gdi32.dll" (ByVal fnPenStyle As Long, ByVal nWidth As Long, ByVal crColor As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 2.0 or later.

Description & Usage

CreatePen creates a pen object. The shape of the pen created by the function is always a square having a side length equal to nWidth. After your program is finished using the pen, it must be deleted via the DeleteObject function.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a handle to the newly created pen.

Visual Basic-Specific Issues

None.

Parameters

fnPenStyle

One of the following flags specifying the style of the pen to create:

- **PS_SOLID**
  - The pen is solid.
- **PS_DASH**
The pen is dashed. \textit{nWidth} must be less than or equal to one.

\textbf{PS_DOT}
- The pen is dotted. \textit{nWidth} must be less than or equal to one.

\textbf{PS_DASHDOT}
- The pen has alternating dashes and dots. \textit{nWidth} must be less than or equal to one.

\textbf{PS_DASHDOTDOT}
- The pen has alternating dashes followed by two dots. \textit{nWidth} must be less than or equal to one.

\textbf{PS_NULL}
- The pen is invisible.

\textbf{PS_INSIDEFRAME}
- The pen is solid. Whenever a drawing function draws a figure inside a bounding rectangle, the dimensions of the figure are shrunk so that the entire figure, including the width of the pen, fits entirely within the bounding rectangle.

\textit{nWidth}
- The width of the pen. If this is 0, the pen is always exactly one pixel wide no matter what.

\textit{crColor}
- The RGB value of the color to give the pen.

### Constant Definitions

\begin{verbatim}
Const PS_SOLID = 0
Const PS_DASH = 1
Const PS_DOT = 2
Const PS_DASHDOT = 3
Const PS_DASHDOTDOT = 4
Const PS_NULL = 5
Const PS_INSIDEFRAME = 6
\end{verbatim}

### Example

```vbscript
' This code is licensed according to the terms and conditions listed here.

' Draw an ellipse on window Form1 using a one-pixel-wide
' square dashed green pen.
Dim hPen As Long ' handle to the pen created
Dim hOldPen As Long ' handle to Form1's previously selected pen
Dim retval As Long ' return value

' Create the square dashed green pen with a width of zero (always one pixel).
hPen = CreatePen(PS_DASH, 0, RGB(0, 255, 0))
' Select the pen for use by window Form1.
hOldPen = SelectObject(Form1.hDC, hPen)

' Draw an ellipse with bounding rectangle (100,150)-(350,300).
retval = Ellipse(Form1.hDC, 100, 150, 350, 300)
```
'Select the old pen for use by Form1.
retval = SelectObject(Form1.hDC, hOldPen)
'
'Delete the pen we created to free up resources.
retval = DeleteObject(hPen)

See Also

CreatePenIndirect

Category

Pens

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 16, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/createpen.html
CreatePenIndirect Function

Declare Function CreatePenIndirect Lib "gdi32.dll" (lpLogPen As LOGPEN) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreatePenIndirect creates a pen object. The pen created is the one which is described by the logical pen information in the structure passed to the function. After your program is finished using the pen, it should be deleted via the DeleteObject function.

Return Value

If an error occured, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a handle to the newly created pen.

Visual Basic-Specific Issues

None.

Parameters

lpLogPen

The description of the attributes to give to the newly created pen.

Example
This code is licensed according to the terms and conditions listed here.

' Draw an ellipse on window Form1 using a one-pixel-wide square dashed green pen.
Dim hPen As Long ' handle to the pen created
Dim hOldPen As Long ' handle to Form1's previously selected pen
Dim lpinfo As LOGPEN ' holds description of the pen
Dim retval As Long ' return value

' Create the square dashed green pen with a width of zero (always one pixel).
lpinfo.lopnStyle = PS_DASH ' dashed line
lpinfo.lopnWidth.x = 0 ' minimum width
lpinfo.lopnWidth.y = 0 ' this member is ignored
lpinfo.lopnColor = RGB(0, 255, 0) ' green
hPen = CreatePenIndirect (lpinfo)
' Select the pen for use by window Form1.
hOldPen = SelectObject (Form1.hDC, hPen)

' Draw an ellipse with bounding rectangle (100,150)-(350,300).
retval = Ellipse (Form1.hDC, 100, 150, 350, 300)

' Select the old pen for use by Form1.
retval = SelectObject (Form1.hDC, hOldPen)
' Delete the pen we created to free up resources.
retval = DeleteObject (hPen)

See Also

CreatePen

Category

Pens

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 16, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
CreatePolygonRgn Function

Declare Function CreatePolygonRgn Lib "gdi32.dll" (lpPoint As POINT_TYPE, ByVal nCount As Long, ByVal nPolyFillMode As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreatePolygonRgn creates a polygonal region and provides a handle to it. The polygon is defined by an array of points specifying its vertices. Note that the polygon fill mode must explicitly be specified, instead of using the one set for whatever device the region is used with. The function returns the handle to the newly created region if successful, or 0 if an error occurred.

lpPoint
An array holding the vertices of the polygonal region. Specify each point in order only once.
nCount
The number of elements in the array passed as lpPoint.
nPolyFillMode
Exactly one of the following flags specifying the polygon fill mode to use for the polygonal region:
ALTERNATE = 1
Alternates between filling and not filling contiguous sections whose boundaries are determined by the edge(s) of the polygon crossing through the polygon's interior.
WINDING = 2
Any section inside the polygon is filled, regardless of any intra-polygonal boundaries and edges.

Example:

' Invert the pixels within a triangular region on window Form1. The triangular region has vertices (150,150), (250, 200), and (100, 200). Note how the points are loaded into the array of vertices.
Dim vertex(0 To 2) As POINT_TYPE  ' array of region's vertices
Dim hrgn As Long  ' handle to the triangular region
Dim retval As Long  ' return value

' Load the vertices of the triangular region into the array.
vertex(0).x = 150: vertex(0).y = 150  ' 1st point: (150,150)
vertex(1).x = 250: vertex(1).y = 200  ' 2nd point: (250,200)
vertex(2).x = 100: vertex(2).y = 200  ' 3rd point: (100,200)

' Create the polygonal region based on the array of vertices.
hrgn = CreatePolygonRgn(vertex(0), 3, ALTERNATE)  ' for a triangle, fill mode is irrelevant
' Invert the pixels within the triangular region on Form1.
retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = \texttt{DeleteObject}(\texttt{hrgn})

\textbf{See Also:} \texttt{CreatePolyPolygonRgn}
\textbf{Category:} Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreatePolyPolygonRgn Function

Declare Function CreatePolyPolygonRgn Lib "gdi32.dll" (lpPoint As POINT_TYPE, lpPolyCounts As Long, ByVal nCount As Long, ByVal nPolyFillMode As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreatePolyPolygonRgn creates a region consisting of multiple polygons. The vertices of all the polygons are passed to the function in the array passed as lpPoint. Another array specifies how many points within that array belong to each polygon. The individual polygons are not joined in any way, forming a region of multiple unconnected polygonal areas. Note that the fill mode for the multi-polygonal region must be specified explicitly, instead of using the filling mode set for whatever device the region is used on. The function returns the handle to the newly created region if successful, or 0 if an error occured.

lpPoint
An array holding the vertices of every polygon making up the new region. Specify each point for each polygon only once. See the example for a demonstration of how to load multiple polygons into this array.

lpPolyCounts
An array specifying how many vertices in the array passed as lpPoint belong to each polygon.

nCount
The number of elements in the array passed as lpPolyCounts.

nPolyFillMode
Exactly one of the following flags specifying the fill mode used for each polygon within the region:
ALTERNATE = 1
  Alternates between filling and not filling contiguous sections whose boundaries are determined by the edge(s) of the polygon crossing through the polygon's interior.
WINDING = 2
  Any section inside the polygon is filled, regardless of any intra-polygonal boundaries and edges.

Example:

' Invert the points lying within a multi-polygonal region on window Form1. The region is made up of a triangle and a diamond. The triangle has vertices (25,25), (50,50), ' and (25,50). The diamond has vertices (150,150), (200,200), (150,250), and (100,200).

Dim vertex(0 To 6) As POINT_TYPE ' holds vertices of each polygon
Dim numvertices(0 To 1) As Long ' holds how many vertices belong to each polygon
Dim hrgn As Long ' handle to the multi-polygonal region
Dim retval As Long ' return value

' Load the vertices of the triangle into the vertex array.
vertex(0).x = 25: vertex(0).y = 25 ' 1st point: (25,25)
vertex(1).x = 50: vertex(1).y = 50 ' 2nd point: (50,50)
```
vertex(2).x = 25: vertex(2).y = 50  ' 3rd point: (25,50)
numvertices(0) = 3  ' three vertices for the triangle

' Load the vertices of the diamond into the vertex array.
vertex(3).x = 150: vertex(3).y = 150  ' 1st point: (150,150)
vertex(4).x = 200: vertex(4).y = 200  ' 2nd point: (200,200)
vertex(5).x = 150: vertex(5).y = 250  ' 3rd point: (150,250)
vertex(6).x = 100: vertex(6).y = 200  ' 4th point: (100,200)
numvertices(1) = 4  ' four vertices for the triangle

' Create the multi-polygonal region and get a handle to it.
hrgn = CreatePolyPolygonRgn(vertex(0), numvertices(0), 2, ALTERNATE)
' Invert the pixels within this region on Form1.
retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = DeleteObject(hrgn)
```

See Also: CreatePolygonRgn
Category: Regions
CreatePopupMenu Function

Declare Function CreatePopupMenu Lib "user32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreatePopupMenu creates a new popup menu object. This menu can then be used for a submenu or a popup menu such as a context menu. The new popup menu is initially empty; use InsertMenuItem to fill it with the desired menu items. When your program no longer needs the popup menu, it should destroy it as necessary using DestroyMenu.

Return Value

If successful, the function returns a handle to the newly created popup menu. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

None.

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreatePopupMenu Lib "user32.dll" () As Long
Public Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type

Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_DEFAULT = &H1000
Public Const MFS_ENABLED = &H0

Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
    MENUITEMINFO) As Long

Public Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type

Public Type TPMPARAMS
    cbSize As Long
    rcExclude As RECT
End Type

Public Declare Function TrackPopupMenuEx Lib "user32.dll" (ByVal hMenu As Long, ByVal _
    fuFlags As Long, ByVal x As Long, ByVal y As Long, ByVal hWnd As Long, lptpm As _
    TPMPARAMS) As Long

Public Const TPM_LEFTALIGN = &H0
Public Const TPM_TOPALIGN = &H0
Public Const TPM_NONOTIFY = &H80
Public Const TPM_RETURNCMD = &H100
Public Const TPM_LEFTBUTTON = &H0

Public Type POINT_TYPE
    x As Long
    y As Long
End Type

Public Declare Function GetCursorPos Lib "user32.dll" (lpPoint As POINT_TYPE) As Long

Public Declare Function SetRectEmpty Lib "user32.dll" (lpRect As RECT) As Long
When the user clicks button Command1, have a very simple popup menu appear. The menu only has two options, divided by a separator bar. The menu is created when needed and is destroyed after its use.

The following application-defined constants are used to name the menu item identifiers used by this example. They are not actually part of the API; instead, they are used just to eliminate "magic numbers."

Private Const ID_ABOUT = 101
Private Const ID_SEPARATOR = 102
Private Const ID_EXIT = 103

Private Sub Command1_Click()
    Dim hPopupMenu As Long    ' handle to the popup menu to display
    Dim mii As MENUITEMINFO   ' describes menu items to add
    Dim tpm As TPMPARAMS      ' identifies the exclusion rectangle
    Dim curpos As POINT_TYPE  ' holds the current mouse coordinates
    Dim menusel As Long       ' ID of what the user selected in the popup menu
    Dim retval As Long        ' generic return value

    ' Create the popup menu that will be displayed.
    hPopupMenu = CreatePopupMenu()
    ' Add the menu's first item: "About This Problem..."
    With mii
        .cbSize = Len(mii)
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        .fType = MFT_STRING
        .fState = MFS_ENABLED Or MFS_DEFAULT
        .wID = ID_ABOUT
        .dwTypeData = "&About This Example..."
        .cch = Len(.dwTypeData)
    End With
    retval = InsertMenuItem(hPopupMenu, 0, 1, mii)
    ' Add the second item: a separator bar.
    With mii
        .fType = MFT_SEPARATOR
        .fState = MFS_ENABLED
        .wID = ID_SEPARATOR
    End With
    retval = InsertMenuItem(hPopupMenu, 1, 1, mii)
    ' Add the final item: "Exit."
    With mii
        .fType = MFT_STRING
        .wID = ID_EXIT
        .dwTypeData = "E&xit"
        .cch = Len(.dwTypeData)
    End With
End Sub
End With
retval = InsertMenuItem(hPopupMenu, 2, 1, mii)

' Determine where the mouse cursor currently is, in order to have
' the popup menu appear at that point.
retval = GetCursorPos(curpos)

' Make the exclusion rectangle empty because there's no need for it here.
With tpm
    ' Size of the structure.
    .cbSize = Len(tpm)
    ' Make the exclusion rectangle empty.
    retval = SetRectEmpty(.rcExclude)
End With

' Display the popup menu at the mouse cursor. Instead of sending messages
' to window Form1, have the function merely return the ID of the user's
' selection.
menusel = TrackPopupMenuEx(hPopupMenu, TPM_TOPALIGN Or TPM_LEFTALIGN Or
TPM_NONOTIFY _
    Or TPM_RETURNCMD Or TPM_LEFTBUTTON, curpos.x, curpos.y, Form1.hWnd, tpm)

' Before acting upon the user's selection, destroy the popup menu now.
retval = DestroyMenu(hPopupMenu)
Select Case menusel
Case ID_ABOUT
    ' Use the Visual Basic MsgBox function to display a short message in
    ' a dialog. Using the MessageBox API function isn't necessary.
    retval = MsgBox("This example demonstrates how to use the API to
display " & _
                "a pop-up menu.", vbOkOnly Or vbInformation, "Windows API
Guide")
Case ID_EXIT
    ' End this program by closing and unloading Form1.
    Unload Form1
End Select
End Sub

See Also

DestroyMenu

Category

Menus

Back to the Function list.
CreateRectRgn Function

Declare Function CreateRectRgn Lib "gdi32.dll" (ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateRectRgn creates a rectangularly-shaped region and provides a handle to it. The rectangle defining the region is specified by passing its upper-left and lower-right corners to the function. Note that the bottom and right edges of the rectangle are not considered to be part of the region. The function returns a handle to the newly created region if successful, or 0 if an error occurred.

X1
   The x-coordinate of the upper-left corner of the rectangle.
Y1
   The y-coordinate of the upper-left corner of the rectangle.
X2
   The x-coordinate of the lower-right corner of the rectangle.
Y2
   The y-coordinate of the lower-right corner of the rectangle.

Example:

' Invert the pixels within a rectangular region on window Form1. The region
' has corners (20,30)-(150,110).
Dim hrgn As Long  ' handle to the rectangular region
Dim retval As Long  ' return value

' Create the rectangular region and get a handle to it.
hrgn = CreateRectRgn(20, 30, 150, 110)  ' has corners (20,30)-(150,110)
' Invert the pixels on Form1 within this region.
retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = DeleteObject(hrgn)

See Also: CreateRectRgnIndirect, CreateRoundRectRgn
Category: Regions
CreateRectRgnIndirect Function

Declare Function CreateRectRgnIndirect Lib "gdi32.dll" (lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateRectRgnIndirect creates a rectangularly-shaped region and provides a handle to it. The rectangle defining the region is specified by the rectangle passed to the function. Note that the bottom and right edges of the rectangle are not considered to be part of the region. The function returns a handle to the newly created region if successful, or 0 if an error occurred.

lpRect
The rectangle which defines the rectangular region to create.

Example:

' Invert the pixels within a rectangular region on window Form1. The region has corners (20,30)-(150,110).
Dim therect As RECT ' rectangle used to create region
Dim hrgn As Long ' handle to the rectangular region
Dim retval As Long ' return value

' Set the rectangle to use to create the region.
retval = SetRect(therect, 20, 30, 150, 110) ' therect = (20,30)-(150,110)
' Create the rectangular region based on this rectangle.
hrgn = CreateRectRgnIndirect(therect)
' Invert the pixels within this region on Form1.
retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
retval = DeleteObject(hrgn)

See Also: CreateRectRgn, CreateRoundRectRgn
Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreateSolidBrush Function

Declare Function CreateSolidBrush Lib "gdi32.dll" (ByVal crColor As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

CreateSolidBrush creates a solid brush object. When used to fill an area or shape, this brush creates a solid fill of a single color. After it is finished using the brush, the program should use DeleteObject to delete the brush and free up system resources. The function returns a handle to the newly created solid brush if successful, or 0 if an error occurred.

crColor

The RGB value of the color to give the solid brush. Visual Basic users can use the intrinsic RGB() function to calculate this value.

Example:

' Draw a rectangle with corners (10,20) and (175,100)
' on window Form1. Use a solid yellow brush to fill the rectangle.
Dim hbrush As Long ' receives handle to the solid yellow brush
Dim holdbrush As Long ' receives handle to Form1's default brush
Dim retval As Long ' return value

hbrush = CreateSolidBrush(RGB(255, 255, 0)) ' create a solid yellow brush
' Save Form1's default brush so we can restore it after the program is finished
holdbrush = SelectObject(Form1.hDC, hbrush) ' select the brush
' Draw the rectangle filled using the solid yellow brush
retval = Rectangle(Form1.hDC, 10, 20, 175, 100)
' Restore Form1's previous brush before destroying the created one
retval = SelectObject(Form1.hDC, holdbrush) ' select old brush
retval = DeleteObject(hbrush) ' destroy the solid yellow brush

See Also: CreateHatchBrush
Category: Brushes

Go back to the alphabetical Function listing.
Go back to the Reference section index.
CreateToolhelp32Snapshot Function

Declare Function CreateToolhelp32Snapshot Lib "kernel32.dll" (ByVal dwFlags As Long, ByVal th32ProcessID As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Description & Usage

CreateToolhelp32Snapshot creates a snapshot of what is running on the computer the moment the function is called. Depending on the flags specified, this snapshot can include running processes and/or threads, among other things. With this snapshot, you can then examine what things were running when the snapshot was made. After your program no longer needs the snapshot, destroy it using CloseHandle.

Return Value

If successful, the function returns a handle to the snapshot that was made. If an error occurred, the function returns -1 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

dwFlags
A combination of the following flags specifying which information to include in the snapshot:

- TH32CS_INHERIT
  Make the returned snapshot handle inheritable.
- TH32CS_SNAPALL
  Include everything (heap list of a process, modules, processes, and threads) in the snapshot.
- TH32CS_SNAPHEAPLIST
  Include the heap list of the process specified by th32ProcessID in the snapshot.
- TH32CS_SNAPMODULE
  Include the module list of the process specified by th32ProcessID in the snapshot.
Include the module list of the process specified by $th32ProcessID$ in the snapshot.

- **TH32CS_SNAPPROCESS**
  - Include the process list in the snapshot.
- **TH32CS_SNAPTHREAD**
  - Include the thread list in the snapshot.

$th32ProcessID$

The identifier of the process for when TH32CS_SNAPHEAPLIST or TH32CS_SNAPMODULE is specified in $dwFlags$. A value of 0 indicates the current process. If neither of those two flags are specified, this parameter is ignored.

### Constant Definitions

- Const TH32CS_INHERIT = &H80000000
- Const TH32CS_SNAPALL = &HF
- Const TH32CS_SNAPHEAPLIST = &H1
- Const TH32CS_SNAPPROCESS = &H2
- Const TH32CS_SNAPTHREAD = &H4
- Const TH32CS_SNAPMODULE = &H8

### Example

Print a list of all the processes currently running on the computer when the user clicks button Command1. To do this, a snapshot of the running process list is taken, and then each process in it is analyzed. The filename of the process and the number of threads owned by it is then displayed. To use this example, place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/c/createtoolhelp32snapshot.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function `CreateToolhelp32Snapshot` Lib "kernel32.dll" (ByVal dwFlags As Long, ByVal th32ProcessID As Long) As Long
Public Const TH32CS_SNAPPROCESS = &H2
Public Type `PROCESSENTRY32`
  dwSize As Long
  cntUsage As Long
  th32ProcessID As Long
  th32DefaultHeapID As Long
  th32ModuleID As Long
  cntThreads As Long
  th32ParentProcessID As Long
  pcPriClassBase As Long
  dwFlags As Long
  szExeFile As String * 260
End Type
Public Declare Function `Process32First` Lib "kernel32.dll" (ByVal hSnapshot As Long, _ lppe As `PROCESSENTRY32`) As Long
Public Declare Function `Process32Next` Lib "kernel32.dll" (ByVal hSnapshot As Long, _ lppe As `PROCESSENTRY32`) As Long
Public Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim hSnapshot As Long  ' handle to the snapshot of the process list
    Dim processInfo As PROCESSENTRY32  ' information about a process in that list
    Dim success As Long    ' success of having gotten info on another process
    Dim exeName As String  ' filename of the process
    Dim retval As Long     ' generic return value

    ' First, make a snapshot of the current process list.
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)

    ' Get information about the first process in the list.
    processInfo.dwSize = Len(processInfo)
    success = Process32First(hSnapshot, processInfo)
    ' Make sure a handle was returned.
    If hSnapshot = -1 Then
        Debug.Print "Unable to take snapshot of process list!"
        Exit Sub
    End If

    ' Loop for each process on the list.
    While success <> 0
        ' Extract the filename of the process (i.e., remove the empty space)
        exeName = Left(processInfo.szExeFile, InStr(processInfo.szExeFile, vbNullChar) - 1)

        ' Display the process name and the number of threads it owns.
        Debug.Print "Process: " & exeName
        Debug.Print "  - Number of threads:" & processInfo.cntThreads

        ' Get information about the next process, if there is one.
        processInfo.dwSize = Len(processInfo)
        success = Process32Next(hSnapshot, processInfo)
    Wend

    ' Destroy the snapshot, now that we no longer need it.
    retval = CloseHandle(hSnapshot)
End Sub

Category

Tool Help

Back to the Function list.
Back to the Reference section.
CreateWindowEx Function

Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, ByVal lParam As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

CreateWindowEx creates a new window. The window can be a "regular" (overlapped) window, a control on another window, or a popup window.

Return Value

If successful, the function returns a handle to the newly created window. If an error occured, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

dwExStyle
A combination of extended styles to give the window.

lpClassName
The name of the class to create the window using. This class must have previously been registered using RegisterClassEx, InitCommonControlsEx, or some other function that registers window classes.

lpWindowName
The name to give the newly created window. For controls, this text will be the initial content of the control. For regular windows, this text appears in the window's title bar.
**dwStyle**
A combination of the window styles to give the window. This is normally a combination of base window styles and styles that are specific to the window’s class.

**x**
The initial x-coordinate of the upper-left corner of the window to create. If this window is a child of another window, this coordinate is relative to its parent window, otherwise it is relative to the screen. If this is CW_USEDEFAULT (for an overlapped window only), y will be ignored and the window will be placed in a default position.

**y**
The initial y-coordinate of the upper-left corner of the window to create. It is interpreted in the same way as x. This parameter is ignored if x is set to CW_USEDEFAULT.

**nWidth**
The initial width of the window. If this is CW_USEDEFAULT (for overlapped windows only), nHeight will be ignored and the window will be given a default size.

**nHeight**
The initial height of the window. If x is set to CW_USEDEFAULT, this parameter will be ignored.

**hWndParent**
A handle to the parent of the window to create. If the window does not have a parent, this should be zero. **Windows 2000**: Set this parameter to HWND_MESSAGE to create a message-only window.

**hMenu**
A handle to a menu to assign to the window. For child windows, this is the child-window identifier, used to notify its parent about events. To not give the window a menu, set this parameter to zero.

**hInstance**
A handle to the instance of the module or program that owns the window.

**lParam**
An additional value to associate with the window, used in the window creation messages.

## Constant Definitions

Const CW_USEDEFAULT = &H80000000
Const HWND_BROADCAST = &HFFFF

## Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSADATA As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
h_name As Long
h_aliases As Long
h_addrtype As Integer
h_length As Integer
h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Type INITCOMMONCONTROLSEX_TYPE
dwSize As Long
dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_ISBLANK = &H469
Public Const IPM_GETADDRESS = &H466
Private hIPControl As Long  ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the
' upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE  ' identifies the control to register
    Dim retval As Long                        ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
        Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the
' address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long   ' the IP address, in host byte order
    Dim ipAddress_n As Long   ' the IP address, in network byte order
    Dim sockinfo As WSADATA   ' information about the Winsock implementation
    Dim pHostinfo As Long     ' pointer to information about the host computer
    Dim hostinfo As HOSTENT   ' information about the host computer
    Dim domainName As String  ' the primary domain name of the host computer
    Dim retvalName As String  ' generic return value

    ' Verify that an IP address was entered.
    retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
    If retval <> 0 Then
        Debug.Print "No IP address was entered!"
        Exit Sub
    End If

    ' Get the IP address entered by the user and verify that all
    ' four fields in the address were entered.
    retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
    If retval < 4 Then
Debug.Print "An incomplete IP address was entered!"
Exit Sub

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)
' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also

DestroyWindow

Category

Windows

Back to the Function list.
Back to the Reference section.
DefWindowProc Function

Declare Function DefWindowProc Lib "user32.dll" Alias "DefWindowProcA" (ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

DefWindowProc explicitly calls the operating system's default window procedure to process a message for a window. This default window procedure provides the minimal functionality necessary for a window procedure function and should be used to provide the default implementation of window messages.

Return Value

The function returns the return value of whatever message was processed.

Visual Basic-Specific Issues

None.

Parameters

- **hWnd**
  A **handle** to the window to process a message for.

- **Msg**
  The message to process.

- **wParam**
  Additional information about the message.

- **lParam**
  Additional information about the message.
Example

' This code is licensed according to the terms and conditions listed here.

' Demonstrate how Visual Basic provides a more robust window
' procedure for windows it creates for the programmer when compared
' to Windows's default window procedure. Do this by "toggling" between
' the default and the Visual Basic provided one.

' *** Place the following code in a module. ***
Public pVBProc As Long ' pointer to Visual Basic's window procedure
' (The above variable defaults to 0 automatically.)

' The following function acts as a wrapper. All it
' does is call the default window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
' Call the default window procedure and return its result.
    WindowProc = DefWindowProc(hWnd, uMsg, wParam, lParam)
End Function

' *** Place the following code wherever you wish. ***
Dim retval As Long ' return value

If pVBProc = 0 Then
    ' Window Form1 is using the VB-provided procedure. Switch to using
    ' the default one and save the pointer to the VB one.
    pVBProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
Else
    ' Window Form1 is using the default procedure (via the wrapper
    ' function). Switch to using the VB one.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pVBProc)
    ' Set pVBProc to 0 so we know which one is being used.
    pVBProc = 0
End If

' By allowing the user to switch back and forth, the differences will
' become apparent.

See Also
CallWindowProc

Category
Window Procedures

Go back to the alphabetical Function listing.
Go back to the Reference section index.
DeleteDC Function

Declare Function DeleteDC Lib "gdi32.dll" (ByVal hdc As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

DeleteDC destroys a device context which was created by CreateDC. Your program should delete a device context once it has finished using it in order to conserve resources. Do not use this function to close a device context gotten from GetDC -- for those, use ReleaseDC instead.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Information

None.

Parameters

hdc

A handle to the device context to delete.

Example

' This code is licensed according to the terms and conditions listed here.

' Print out a page with an ellipse drawn with a thickened black pen on it. The page is printed on the computer's default printer.
The following are special declarations needed to allow string manipulation functions to use pointers to strings.

Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long

Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

Variable declarations

Dim hPrintDC As Long  ' handle to the printer's device context
Dim di As DOCINFO  ' information about the document to print
Dim hPen As Long  ' handle to the pen to draw the ellipse with
Dim hOldPen As Long  ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long  ' 3076-byte buffer
Dim pi2 As PRINTER_INFO_2  ' receives info about the default printer
Dim printret As Long  ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long  ' receives space requires for EnumPrinters
Dim retval As Long  ' return value

Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.

    retval = EnumPrinters(PRINTER_ENUM_DEFAULT, "", 2, buffer(0), 3076, spaceneeded, printret)
    If retval = 0 Then
        Debug.Print "No default printer is configured."
        End ' abort the program
    End If

    Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.
    pi2.pPrinterName = Space(lstrlen(buffer(1)))
    retval = lstrcpy(pi2.pPrinterName, buffer(1))
    pi2.pDriverName = Space(lstrlen(buffer(4)))
    retval = lstrcpy(pi2.pDriverName, buffer(4))

Create a device context to the printer, using its default initialization.

    hPrintDC = CreateDC("", pi2.pPrinterName, 0, ByVal CLng(0))

Create a solid black brush with a thickness of 5.

    hPen = CreatePen(PS_SOLID, 5, RGB(0, 0, 0))

Load information about the document to print into the structure.

    di.cbSize = Len(di)  ' size of structure
    di.lpszDocName = "Printer API Demonstration"  ' name of document
    di.lpszOutput = 0  ' do not print to a file
    di.lpszDatatype = ""  ' data type of file doesn't apply
    di-fwType = 0  ' no additional information

Begin the print job.

    retval = StartDoc(hPrintDC, di)

Begin the first and only page to print.

    retval = StartPage(hPrintDC)

Select the pen for use with the printer.

hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

CreateDC

Category

Devices

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 6, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/d/deletedc.html
DeleteFile Function

Declare Function DeleteFile Lib "kernel32.dll" Alias "DeleteFileA"
(ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

DeleteFile deletes a file completely -- it does not send it to the Recycle Bin. It also doesn't prompt to confirm the deletion, so use it carefully. The function returns 1 if successful, or 0 if an error occurred (most likely the file doesn't exist).

lpFileName
The name of the file to delete.

Example:

' Delete the file C:\Dummy\thefile.txt
Dim retval As Long   ' return value

retval = DeleteFile("C:\Dummy\thefile.txt")
If retval = 1 Then Debug.Print "File deleted successfully."

Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
DeleteObject Function

Declare Function DeleteObject Lib "gdi32.dll" (ByVal hObject As Long) As Long

DeleteObject deletes an object after the program has finished using it. These objects include bitmaps, brushes, fonts, palettes, pens, and regions. Of course the object should never be deleted until it is no longer in use by any devices (see the example for a demonstration). Deleting the object frees up system resources associated with it. The function returns 1 if successful, or 0 if an error occurred.

hObject
   A handle to the bitmap, brush, font, palette, pen, or region to delete.

Example:

' Draw a rectangle with corners (10,20) and (175,100)
' on window Form1. Use a solid yellow brush to fill the rectangle.
Dim hbrush As Long   ' receives handle to the solid yellow brush
Dim holdbrush As Long   ' receives handle to Form1's default brush
Dim retval As Long   ' return value

hbrush = CreateSolidBrush(RGB(255, 255, 0))   ' create a solid yellow brush
' Save Form1's default brush so we can restore it after the program is finished
holdbrush = SelectObject(Form1.hDC, hbrush)   ' select the brush
' Draw the rectangle filled using the solid yellow brush
retval = Rectangle(Form1.hDC, 10, 20, 175, 100)
' Restore Form1's previous brush before destroying the created one
retval = SelectObject(Form1.hDC, holdbrush)   ' select old brush
retval = DeleteObject(hbrush)   ' destroy the solid yellow brush

Category: Devices
This page is at http://www.vbapi.com/ref/d/deleteobject.html
DestroyCursor Function

Declare Function DestroyCursor Lib "user32.dll" (ByVal hCursor As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

DestroyCursor destroys a cursor and deletes its handle. The cursor must have been created using the CreateCursor functions. Do not use this function with a cursor handle obtained in any other manner! The function returns 1 if successful, or 0 if an error occured.

hCursor
A handle to the cursor to delete.

Example:

' Create a 32x32 color cursor shaped somewhat like a yin-yang symbol.
' (The bit masks come from Microsoft's documentation on the API cursors function, just to
' give them their due credit.) Note how the masks are loaded into the arrays. The new
' cursor is then set to be the cursor for 10 seconds.
Dim hnewcursor As Long ' newly created cursor
Dim holdcursor As Long ' receives handle of default cursor
Dim andbuffer As String, xorbuffer As String ' buffers for masks
Dim andbits(0 To 127) As Byte ' stores the AND mask
Dim xorbits(0 To 127) As Byte ' stores the XOR mask
Dim c As Integer, retval As Long ' counter and return value

' Unfortunately, VB does not provide a nice way to load lots of information into an
' array. To load the AND and XOR masks, we put the raw hex values into the string buffers
' and use a loop to convert the hex values into numeric values and load them into
' the elements of the array. Yes, it's ugly, but there's no better way. Note the
' use of the line-continuation character here. Each sequence of eight hex
' characters represents one line in the 32x32 cursor.
andbuffer = "FFFC3FFF" & "FFC01FFF" & "FF003FFF" & "FE000FFF" & _
"F701FFFF" & "F003FFFF" & "F003FFFF" & "E007FFFF" & _
"C007FFFF" & "C000FFFF" & "80000000" & "80000000" & _
"80000000" & "80000000" & "0003FFFF" & "0000FFFF" & _
"00007FFF" & "00001FFF" & "00000000" & "80000000" & _
"80000000" & "80000000" & "C0000000" & "C0000000" & _
"E0000000" & "F0001FFF" & "F0001FFF" & "F8003FFF" & _
"FE007FFF" & "FF000FFF" & "FFC3FFFF" & "FFFFFFFF"
xorbuffer = "00000000" & "0003C000" & "003F0000" & "00FE0000" & _
"0EFC0000" & "07F80000" & "07F80000" & "0FF00000" & _
"1FF00000" & "1FE00000" & "3FE00000" & "3FE00000" & _
"3FF00000" & "7FF00000" & "7FF80000" & "7FFC00000" & _
"7FFF0000" & "7FFFF8000" & "7FFFFFE0000" & "3FFFFE0000" & _
"3FC7F0000" & "3F83F0000" & "1F83F0000" & "1F83F0000" & _
"0FC7E0000" & "07FFC0000" & "07FFFE0000" & "01FF80000" & _
"0FF00000" & "003C0000" & "00000000" & "00000000"

' Now load these hex values into the proper arrays.
For c = 0 To 127
    andbits(c) = "&H" & Mid(andbuffer, 2 * c + 1)
    xorbits(c) = "&H" & Mid(xorbuffer, 2 * c + 1)
Next c

' Finally, create this cursor! The hotspot is at (19,2) on the cursor.
hnewcursor = CreateCursor(App.hInstance, 19, 2, 32, 32, andbits(0), xorbits(0))

' Set the new cursor as the current cursor for 10 seconds and then switch back.
holdcursor = SetCursor(hnewcursor)  ' change cursor
Sleep 10000  ' wait for 10 seconds
retval = SetCursor(holdcursor)  ' change cursor back

' Destroy the new cursor.
retval = DestroyCursor(hnewcursor)

See Also: CreateCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
DestroyIcon Function

Declare Function DestroyIcon Lib "user32.dll" (ByVal hIcon As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**DestroyIcon** destroys an icon and frees the memory which contained the icon. Some functions which provide an icon (sometimes but not necessarily creating it) require programs to use this function after using an icon, while others do not. Check the icon function in question to determine whether or not the icon must be destroyed. An icon cannot be in use when the program destroys it, whether it is used by the program or other programs.

Return Value

If an error occurred, the function returns 0 (use **GetLastError** to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

**hIcon**

A **handle** to the icon to destroy.

Example
' This code is licensed according to the terms and conditions listed here.

' Display the first icon (index 0) stored in the executable file C:\MyApp\Prog.exe on window Form1. The icon must be destroyed after the program finishes using it.
Dim hIcon As Long ' handle to the function gotten from the executable file
Dim retval As Long ' return value

' Extract the first icon stored in the aforementioned executable file.
hIcon = ExtractIcon(App.hInstance, "C:\MyApp\Prog.exe", 0)
' Only attempt to display the icon if we successfully extracted it.
If hIcon = 0 Then
    Debug.Print "Failed to extract the icon -- aborting."
    End ' terminate the program
Else
    ' Display the icon at coordinates (100, 75) on window Form1.
    retval = DrawIcon(Form1.hDC, 100, 75, hIcon)
    ' Although the icon's image is still visible, the icon itself is not in use.
    ' Therefore we destroy it to free up resources.
    retval = DestroyIcon(hIcon)
End If

Category

Icons

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 4, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/d/destroyicon.html
DestroyMenu Function

Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

DestroyMenu destroys a menu resource. This menu can be either a "regular" menu (i.e., a menu bar) or a popup menu. Menus should be destroyed when no longer needed in order to free resources. However, it is not necessary to call DestroyMenu to destroy a menu that is assigned as a window's menu (or any submenus of that menu). Those menus are automatically destroyed when their window closes.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hMenu
A handle to the menu to be destroyed.

Example

' This code is licensed according to the terms and conditions listed here.
' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreatePopupMenu Lib "user32.dll" () As Long
Public Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_DEFAULT = &H1000
Public Const MFS_ENABLED = &H0
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
    MENUITEMINFO) As Long
Public Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type
Public Type TPMPARAMS
    cbSize As Long
    rcExclude As RECT
End Type
Public Declare Function TrackPopupMenuEx Lib "user32.dll" (ByVal hMenu As Long, ByVal _
    fuFlags As Long, ByVal x As Long, ByVal y As Long, ByVal hWnd As Long, lptpm As _
    TPMPARAMS) As Long
Public Const TPM_LEFTALIGN = &H0
Public Const TPM_TOPALIGN = &H0
Public Const TPM_NONOTIFY = &H80
Public Const TPM_RETURNCMD = &H100
Public Const TPM_LEFTBUTTON = &H0
Public Type POINT_TYPE
    x As Long
    y As Long
End Type
Public Declare Function GetCursorPos Lib "user32.dll" (lpPoint As POINT_TYPE) As Long
Public Declare Function `SetRectEmpty Lib "user32.dll" (lpRect As RECT) As Long

' When the user clicks button Command1, have a very simple popup menu appear. The
' menu only has two options, divided by a separator bar. The menu is created when
' needed and is destroyed after its use.

' The following application-defined constants are used to name the menu item
' identifiers used by this example. They are not actually part of the API; instead,
' they are
' used just to eliminate "magic numbers."
Private Const ID_ABOUT = 101
Private Const ID_SEPARATOR = 102
Private Const ID_EXIT = 103

Private Sub Command1_Click()
    Dim hPopupMenu As Long ' handle to the popup menu to display
    Dim mii As MENUITEMINFO ' describes menu items to add
    Dim tpm As TPMPARAMS      ' identifies the exclusion rectangle
    Dim curpos As POINT_TYPE  ' holds the current mouse coordinates
    Dim menusel As Long       ' ID of what the user selected in the popup menu
    Dim retval As Long        ' generic return value

    ' Create the popup menu that will be displayed.
    hPopupMenu = CreatePopupMenu()

    ' Add the menu's first item: "About This Problem..."
    With mii
        .cbSize = Len(mii)
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        .fType = MFT_STRING
        .wID = ID_ABOUT
        .fState = MFS_ENABLED Or MFS_DEFAULT
        .dwTypeData = "&About This Example..."
        .cch = Len(.dwTypeData)
    End With
    retval = InsertMenuItem(hPopupMenu, 0, 1, mii)

    ' Add the second item: a separator bar.
    With mii
        .fType = MFT_SEPARATOR
        .fState = MFS_ENABLED
        .wID = ID_SEPARATOR
    End With
    retval = InsertMenuItem(hPopupMenu, 1, 1, mii)

    ' Add the final item: "Exit".
    With mii
        .fType = MFT_STRING
        .wID = ID_EXIT
    End With
End Sub
Windows API Guide: DestroyMenu Function

```
    .dwTypeData = "E&xit"
    .cch = Len(.dwTypeData)
End With
    retval = InsertMenuItem(hPopupMenu, 2, 1, mii)
    
    ' Determine where the mouse cursor currently is, in order to have
    ' the popup menu appear at that point.
    retval = GetCursorPos(curpos)
    
    ' Make the exclusion rectangle empty because there's no need for it here.
    With tpm
        ' Size of the structure.
        .cbSize = Len(tpm)
        ' Make the exclusion rectangle empty.
        retval = SetRectEmpty(.rcExclude)
    End With
    
    ' Display the popup menu at the mouse cursor. Instead of sending messages
    ' to window Form1, have the function merely return the ID of the user's
    ' selection.
    menusel = TrackPopupMenuEx(hPopupMenu, TPM_TOPALIGN Or TPM_LEFTALIGN Or
    TPM_NONOTIFY Or TPM_RETURNCMD Or TPM_LEFTBUTTON, curpos.x, curpos.y, Form1.hWnd, tpm)
    
    ' Before acting upon the user's selection, destroy the popup menu now.
    retval = DestroyMenu(hPopupMenu)
    Select Case menusel
        Case ID_ABOUT
            ' Use the Visual Basic MsgBox function to display a short message in
            ' box. Using the MessageBox API function isn't necessary.
            retval = MsgBox("This example demonstrates how to use the API to display " & _
                            "a pop-up menu.", vbOkOnly Or vbInformation, "Windows API Guide")
        Case ID_EXIT
            ' End this program by closing and unloading Form1.
            Unload Form1
    End Select
End Sub
```

See Also

CreatePopupMenu

Category

Menus
EnableWindow Function

Declare Function EnableWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal fEnable As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

EnableWindow enables or disables a window. If a window is disabled, it cannot receive the focus and will ignore any attempted input. Some types of windows, such as buttons and other controls, will appear grayed when disabled, although any window can be enabled or disabled. The function returns 0 if the window had previously been enabled, or a non-zero value if the window had been disabled.

hwnd
A handle to the window to enable or disable.

fEnable
If 0, the window will be disabled. If non-zero, the window will be enabled.

Example:

' Reverse the enabled status of window Command1. If the window is disabled, enable it; if it is enabled, disable it.
Dim wasenabled As Long ' receives enabled/disabled status of Command1
Dim retval As Long ' return value

' Determine if the window Command1 is currently enabled or not.
wasenabled = IsWindowEnabled(Command1 hWnd)
If wasenabled = 0 Then ' if not enabled, enable it
    retval = EnableWindow(Command1 hWnd, 1)
Else ' if enabled, disable it
    retval = EnableWindow(Command1 hWnd, 0)
End If

See Also: IsWindowEnabled
Category: Windows
Windows API Guide: EnableWindow Function

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/enablewindow.html
DestroyWindow Function

Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

DestroyWindow destroys a window. Any menus, timers, or child windows are automatically destroyed along with it. DestroyWindow only works with windows that are owned by the calling program, however.

Return Value

If successful, the function returns a nonzero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to destroy.

Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Type **WSADATA**
  wVersion As Integer
  wHighVersion As Integer
  szDescription As String * 257
  szSystemStatus As String * 129
  iMaxSockets As Long
  iMaxUdpDg As Long
  lpVendorInfo As Long
End Type

Public Declare Function **WSAStartup** Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As **WSADATA**) As Long

Public Declare Function **WSACleanup** Lib "wsock32.dll" () As Long

Public Type **HOSTENT**
  h_name As Long
  h_aliases As Long
  h_addrtype As Integer
  h_length As Integer
  h_addr_list As Long
End Type

Public Const AF_INET = 2

Public Declare Function **htonl** Lib "wsock32.dll" (ByVal hostlong As Long) As Long

Public Declare Function **gethostbyaddr** Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long

Public Declare Sub **CopyMemory** Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)

Public Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long

Public Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

Public Type **INITCOMMONCONTROLSEX_TYPE**
  dwSize As Long
  dwICC As Long
End Type

Public Declare Function **InitCommonControlsEx** Lib "comctl32.dll" (lpInitCtrls As **INITCOMMONCONTROLSEX_TYPE**) As Long

Public Const ICC_INTERNET_CLASSES = &H800

Public Declare Function **CreateWindowEx** Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hWndMenu As Long) As Long
Windows API Guide: DestroyWindow Function

Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long

Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000

Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
    Dim retval As Long                        ' generic return value
    
    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)
    
    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",
        WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
        Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = DestroyWindow(hIPControl)
End Sub

Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long   ' the IP address, in host byte order
    Dim ipAddress_n As Long   ' the IP address, in network byte order
    Dim sockinfo As WSADATA   ' information about the Winsock implementation
    Dim pHostinfo As Long     ' pointer to information about the host computer
    Dim hostinfo As HOSTENT   ' information about the host computer

    ' Look up the primary domain name of the host computer identified by the
    ' address in the IP Address control.
Dim domainName As String  ' the primary domain name of the host computer
Dim retval As Long       ' generic return value

' Verify that an IP address was entered.
retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
If retval <> 0 Then
    Debug.Print "No IP address was entered!"
    Exit Sub
End If

' Get the IP address entered by the user and verify that all four fields in the address were entered.
retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
If retval < 4 Then
    Debug.Print "An incomplete IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)

' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also

CreateWindowEx, WM_CLOSE

Category
DrawIcon Function

Declare Function DrawIcon Lib "user32.dll" (ByVal hDC As Long, ByVal x As Long, ByVal y As Long, ByVal hIcon As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**DrawIcon** displays an icon on a device. The icon's position is determined by a coordinate pair passed to the function identifying the coordinates of the upper-left corner of the icon. The icon is always drawn in its normal dimensions.

Return Value

If an error occurred, the function returns 0 (use **GetLastError** to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hDC**
  
  A handle to a device context to the device to draw the icon on.

- **x**
  
  The x-coordinate of the point to position the upper-left corner of the icon's image at.

- **y**
  
  The y-coordinate of the point to position the upper-left corner of the icon's image at.

- **hIcon**
A handle to the icon to display.

Example

' This code is licensed according to the terms and conditions listed here.

' Display the first icon (index 0) stored in the executable file
' C:\MyApp\Prog.exe on window Form1. The icon must be destroyed after the
' program finishes using it.
Dim hIcon As Long  ' handle to the function gotten from the executable file
Dim retval As Long  ' return value

' Extract the first icon stored in the aforementioned executable file.
hIcon = ExtractIcon(App.hInstance, "C:\MyApp\Prog.exe", 0)
' Only attempt to display the icon if we successfully extracted it.
If hIcon = 0 Then
    Debug.Print "Failed to extract the icon -- aborting."
    End  ' terminate the program
Else
    ' Display the icon at coordinates (100, 75) on window Form1.
    retval = DrawIcon(Form1.hDC, 100, 75, hIcon)
    ' Although the icon's image is still visible, the icon itself is not in use.
    ' Therefore we destroy it to free up resources.
    retval = DestroyIcon(hIcon)
End If

See Also

DrawIconEx

Category

Icons

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 4, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
Windows API Guide: DrawIcon Function

This page is at http://www.vbapi.com/ref/d/drawicon.html
DrawIconEx Function

Declare Function DrawIconEx Lib "user32.dll" (ByVal hdc As Long, ByVal xLeft As Long, ByVal yTop As Long, ByVal hIcon As Long, ByVal cxWidth As Long, ByVal cyWidth As Long, ByVal istepIfAniCur As Long, ByVal hbrFlickerFreeDraw As Long, ByVal diFlags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

DrawIconEx displays either an icon or a cursor (or a single frame of an animated cursor) on a device. The image's position is determined by passing the coordinates of the upper-left corner of the image. The function can stretch the image in either direction as well as specify other display parameters.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hdc**: A handle to a device context to the device to draw the icon or cursor on.
- **xLeft**: The x-coordinate of the point to position the upper-left corner of the icon's or cursor's image at.
- **yTop**: The y-coordinate of the point to position the upper-left corner of the icon's or cursor's image at.
- **hIcon**:
A handle to the icon or cursor to draw.

\textit{cxWidth}

The width in pixels to stretch the icon's or cursor's image to. If this is 0, the image is drawn using either the icon's or cursor's width or the default width of the system, depending on the flags passed as \textit{diFlags}.

\textit{cyWidth}

The height in pixels to stretch the icon's or cursor's image to. If this is 0, the image is drawn using either the icon's or cursor's height or the default height of the system, depending on the flags passed as \textit{diFlags}.

\textit{istepIfAniCursor}

If \textit{hIcon} is a handle to an animated cursor, this specifies the index of the particular frame to draw. Otherwise, this parameter is ignored.

\textit{hbrFlickerFreeDraw}

A handle to the \textbf{brush} to use as the icon's or cursor's background. The background is added to the icon's or cursor's image using a flicker-free method. If this parameter is 0, the image is drawn directly onto the device without first adding a background.

\textit{diFlags}

A combination of the following flags specifying how to draw the icon or cursor:

\textbf{DL_COMPAT}

Draw the icon or cursor using the system default image instead of the user-specified image.

\textbf{DL_DEFAULTSIZE}

If \textit{cxWidth} and \textit{cyWidth} are set to 0, draw the icon or cursor using the height and width settings defined by the system metrics. If this flag is not specified and the two parameters are set to 0, the icon or cursor is drawn using its own dimensions.

\textbf{DL_IMAGE}

Draw the icon's or cursor's image data onto the device.

\textbf{DL_MASK}

Draw the icon's or cursor's mask data onto the device.

\textbf{DL_NORMAL}

Draw the icon or cursor using both its image and mask, as usual.

\section*{Constant Definitions}

\begin{verbatim}
Const DI_COMPAT = &H4
Const DI_DEFAULTSIZE = &H8
Const DI_IMAGE = &H2
Const DI_MASK = &H1
Const DI_NORMAL = &H3
\end{verbatim}

\section*{Example}

\begin{verbatim}
' This code is licensed according to the terms and conditions listed here.

' Extract all of the regular-sized icons from the file
' C:\MyApp\Prog.exe. Display them in a row, stretching or shrinking them to
' a width of 32 and a height of 64. Note how dynamically allocated arrays
' are used to receive the icon handles. Draw all icons on a light-gray
' background on the window Form1.
Dim hIcons() As Long  ' dynamic array to receive handles to the icons
Dim numicons As Long  ' number of regular icons in the file
Dim hBrush As Long    ' handle to the background brush to use
\end{verbatim}
Dim c As Long  ' counter variable
Dim retval As Long  ' return value

' Determine how many regular icons exist in the file and resize
' the array accordingly.
numicons = ExtractIconEx("C:\MyApp\Prog.exe", -1, ByVal 0, ByVal 0, 0)
If numicons = 0 Then
    Debug.Print "No icons found in the file -- aborting."
    End ' abort the program if failure occurs
End If
ReDim hIcons(0 To numicons - 1) As Long  ' resize the array to hold all the handles
' Get a handle to the stock solid light gray brush to use for the background.
hBrush = GetStockObject(LTGRAY_BRUSH)  ' handle to the brush

' Extract all of the icons to display.
retval = ExtractIconEx("C:\MyApp\Prog.exe", numicons, hIcons(0), ByVal 0, 0)

' Loop through each icon, displaying it as previously mentioned.
For c = 0 To numicons - 1
    ' The x coordinate equals 32 * c. The y coordinate is always 0.
    ' Display this particular icon.
    retval = DrawIconEx(Form1.hDC, 32 * c, 0, hIcons(c), 32, 64, 0, hBrush, DI_NORMAL)
    ' Now destroy this icon since we no longer are using it.
    retval = DestroyIcon(hIcons(c))
Next c

See Also

DrawIcon

Category

Icons

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 5, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/d/drawiconex.html
Ellipse Function

Declare Function Ellipse Lib "gdi32.dll" (ByVal hdc As Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long

Platforms: Win32s, Win 95/98, Win NT

Ellipse draws an ellipse on a device. The two coordinate pairs passed to the function are not part of the ellipse itself, but define its bounding rectangle. The bounding rectangle is the smallest possible rectangle containing the ellipse. The ellipse is drawn using the device's current drawing color and is filled using the current filling color and brush, if any. The function returns 0 if it fails, or 1 if it succeeds.

hdc
The device context of the object to draw on.

X1
The x coordinate of the bounding rectangle's upper-left corner.

Y1
The y coordinate of the bounding rectangle's upper-left corner.

X2
The x coordinate of the bounding rectangle's lower-right corner.

Y2
The y coordinate of the bounding rectangle's lower-right corner.

Example:

' Draw a red ellipse with bounding rectangle (25,30)-(100,75)
' on PictureBox1
Dim retval As Long ' return value
PictureBox1.ForeColor = RGB(255, 0, 0) ' set color to draw in to red
retval = Ellipse(PictureBox1.hdc, 25, 30, 100, 75)

See Also: AngleArc, Arc, ArcTo, Chord, Pie
Category: Filled Shapes
EndDoc Function

Declare Function EndDoc Lib "gdi32.dll" (ByVal hdc As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 2.0 or later.

Description & Usage

**EndDoc** informs the print spooler that the program has finished providing information for how to print a document. This function tells the spooler to finally print the document. The **StartDoc** and **EndDoc** functions must bracket all the code which draws the document on the printer (see the example for an illustration).

Return Value

If an error occurred, the function returns either zero or a negative value (**Windows NT, 2000**: use **GetLastError** to get the error code). If successful, the function returns a positive value.

Visual Basic-Specific Issues

None.

Parameters

**hdc**

A handle to a device context to the printer which the document is to be printed using.

Example

' This code is licensed according to the terms and conditions listed here.

' Print out a page with an ellipse drawn with a thickened black pen on it. The page is printed on the computer's default printer.
The following are special declarations needed to allow string manipulation functions to use pointers to strings.

Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

' Variable declarations
Dim hPrintDC As Long ' handle to the printer's device context
Dim di As **DOCINFO** ' information about the document to print
Dim hPen As Long ' handle to the pen to draw the ellipse with
Dim hOldPen As Long ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long ' 3076-byte buffer
Dim pi2 As **PRINTER_INFO_2** ' receives info about the default printer
Dim printret As Long ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long ' receives space requires for EnumPrinters
Dim retval As Long ' return value

' Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.
retval = **EnumPrinters** (PRINTER_ENUM_DEFAULT, "", 2, buffer(0), 3076, spaceneeded, printret)
If retval = 0 Then
    Debug.Print "No default printer is configured."
    End ' abort the program
End If

' Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.
pi2.pPrinterName = Space(lstrlen(buffer(1)))
retval = **lstrcpy**(pi2.pPrinterName, buffer(1))
pi2.pDriverName = Space(lstrlen(buffer(4)))
retval = **lstrcpy**(pi2.pDriverName, buffer(4))

' Create a device context to the printer, using its default initialization.
hPrintDC = **CreateDC**("", pi2.pPrinterName, 0, ByVal CLng(0))
' Create a solid black brush with a thickness of 5.
hPen = **CreatePen**(PS_SOLID, 5, RGB(0, 0, 0))

' Load information about the document to print into the structure.
di.cbSize = Len(di) ' size of structure
di.lpszDocName = "Printer API Demonstration" ' name of document
di.lpszOutput = 0 ' do not print to a file
di.lpszDatatype = "" ' data type of file doesn't apply
di-fwType = 0 ' no additional information

' Begin the print job.
retval = **StartDoc**(hPrintDC, di)
' Begin the first and only page to print.
retval = **StartPage**(hPrintDC)
' Select the pen for use with the printer.
hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

StartDoc

Category

Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 2, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/enddoc.html
EndPage Function

Declare Function EndPage Lib "gdi32.dll" (ByVal hDC As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 2.0 or later.

Description & Usage

*EndPage* informs the print spooler that the program has finished drawing the contents of a printed page. The *StartPage* and *EndPage* functions must bracket all code which draws the contents of the printed page. The print spooler must have already been informed that it is receiving a document to be printed via *StartDoc* before drawing the page had begun. The printer does not actually print the page until the print spooler is told via *EndDoc* that the entire document is complete.

Return Value

If an error occurred, the function returns either zero or a negative value (*Windows NT, 2000*: use *GetLastError* to get the error code). If successful, the function returns a positive value.

Visual Basic-Specific Issues

None.

Parameters

* hDC
  
  A [handle](https://docs.microsoft.com/en-us/windows/win32/winProgAPI/handle-api-reference) to a device context to the printer being used to print the document.

Example

' This code is licensed according to the terms and conditions listed [here](http://www.vbworld.com/vbapi/ref/e/endpage.html).

' Print out a page with an ellipse drawn with a thickened black pen on it. The page is printed on the computer's default printer.
The following are special declarations needed to allow string manipulation functions to use pointers to strings.

```vbscript
Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long
```

Variable declarations:

```vbscript
Dim hPrintDC As Long  ' handle to the printer's device context
Dim di As DOCINFO  ' information about the document to print
Dim hPen As Long  ' handle to the pen to draw the ellipse with
Dim hOldPen As Long  ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long  ' 3076-byte buffer
Dim pi2 As PRINTER_INFO_2  ' receives info about the default printer
Dim printret As Long  ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long  ' receives space requires for EnumPrinters
Dim retval As Long  ' return value
```

Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.

```vbscript
retval = EnumPrinters(PRINTER_ENUM_DEFAULT, ",", 2, buffer(0), 3076, spaceneeded, printret)
If retval = 0 Then
    Debug.Print "No default printer is configured."
    End  ' abort the program
End If
```

Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.

```vbscript
pi2.pPrinterName = Space(lstrlen(buffer(1)))
retval = lstrcpy(pi2.pPrinterName, buffer(1))
pi2.pDriverName = Space(lstrlen(buffer(4)))
retval = lstrcpy(pi2.pDriverName, buffer(4))
```

Create a device context to the printer, using its default initialization.

```vbscript
hPrintDC = CreateDC("", pi2.pPrinterName, 0, ByVal CLng(0))
```

Create a solid black brush with a thickness of 5.

```vbscript
hPen = CreatePen(PS_SOLID, 5, RGB(0, 0, 0))
```

Load information about the document to print into the structure.

```vbscript
di.cbSize = Len(di)  ' size of structure
di.lpszDocName = "Printer API Demonstration"  ' name of document
di.lpszOutput = 0  ' do not print to a file
di.lpszDatatype = ""  ' data type of file doesn't apply
di-fwType = 0  ' no additional information
```

Begin the print job.

```vbscript
retval = StartDoc(hPrintDC, di)
```

Begin the first and only page to print.

```vbscript
retval = StartPage(hPrintDC)
```

Select the pen for use with the printer.
hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

StartPage

Category

Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 5, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information
Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/endpage.html
EnumChildWindows Function

Declare Function EnumChildWindows Lib "user32.dll" (ByVal hWndParent As Long, ByVal lpEnumFunc As Long, ByVal lParam As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

EnumChildWindows enumerates and provides handles to all of the child windows of a given window. This function will also enumerate any children of the child windows. Each time a child window is located, the function passes that handle to a program-defined callback function. The function continues doing so until all child windows have been enumerated, or until the process has been aborted.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

 hWndParent
  A handle to the parent window to enumerate the child windows of.

 lpEnumFunc
  A pointer to the application-defined callback function EnumChildProc.

 lParam
  An additional value to pass to the application-defined callback function.

Example:
Display the window title of all children of window MDIForm1. This task is given to the callback function, which will receive each handle individually.

*** Place this code in a module. This is the callback function. ***
This function displays the title bar text of the window identified by hwnd.

Public Function EnumChildProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
Dim slength As Long, wintext As String ' window title text length and buffer
Dim retval As Long ' return value
Static winnum As Integer ' counter keeps track of how many windows have been enumerated

    winnum = winnum + 1 ' one more window enumerated....
    slength = GetWindowTextLength(hwnd) + 1 ' get length of title bar text
    buffer = Space(slength) ' make room in the buffer
    retval = GetWindowText(hwnd, buffer, slength) ' get title bar text
    Debug.Print "Window "; winnum; " : "; ' display number of enumerated window
    Debug.Print Left(buffer, slength - 1) ' display title bar text of enumerated window

    EnumChildProc = 1 ' nonzero return value means continue enumeration
End Function

*** Place this code wherever you want to enumerate the windows. ***
Dim retval As Long ' return value

' Use the above callback function to list all of the enumerated windows. Note that lParam is ' set to 0 because we don't need to pass any additional information to the function.
retval = EnumChildWindows(MDIForm1.hWnd, AddressOf EnumChildProc, 0)

See Also
EnumThreadWindows, EnumWindows

Category
Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 15, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
EnumDisplaySettings Function

Declare Function EnumDisplaySettings Lib "user32.dll" Alias "EnumDisplaySettingsA" (ByVal lpszDeviceName As String, ByVal iModeNum As Long, lpDevMode As DEVMODE) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.51 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

`EnumDisplaySettings` gets the display settings of one of a display device's graphics modes. Specifically, the function gets the display mode's resolution, color depth, and frequency, along with some additional information.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (Windows NT/2000: use `GetLastError` to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- `lpszDeviceName`
  - **Windows 95, 98**: This must be a null string. **Windows NT, 2000**: If this is a null string, the current display device is used. Otherwise, this is the device name of the display device to examine. The string is of the form "\\DisplayX", where X is 1, 2, or 3.
- `iModeNum`
The number of the graphics mode to retrieve information about. This could also be one of the following flags specifying a graphics mode:

- **ENUM_CURRENT_SETTINGS**
  - Get information about the current display settings.
- **ENUM_REGISTRY_SETTINGS**
  - Get information about the display settings stored in the registry.

The function `lpDevMode` receives information about the graphics mode. Only the `dmBitsPerPixel`, `dmPelsWidth`, `dmPelsHeight`, `dmDisplayFlags`, and `dmDisplayFrequency` members have useful data in them. Before calling the function, this structure's `dmSize` member must be properly set.

**Constant Definitions**

```vbnet
Const ENUM_CURRENT_SETTINGS = -1
Const ENUM_REGISTRY_SETTINGS = -2
```

**Example**

Display information about the current display settings for the monitor. To use this example, place a command button named Command1 on a form window.

```vbnet
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type DEVMODE
    dmDeviceName As String * 32
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer
    dmPaperSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
```

dmFormName As String * 32
dmUnusedPadding As Integer
dmBitsPerPixel As Integer
dmPelsWidth As Long
dmPelsHeight As Long
dmDisplayFlags As Long
dmDisplayFrequency As Long
' The following only appear in Windows 95, 98, 2000
dmICMMethod As Long
dmICMIntent As Long
dmMediaType As Long
dmDitherType As Long
dmReserved1 As Long
dmReserved2 As Long
' The following only appear in Windows 2000
dmPanningWidth As Long
dmPanningHeight As Long

End Type

Public Declare Function EnumDisplaySettings Lib "user32.dll" Alias "EnumDisplaySettingsA" (ByVal lpszDeviceName As String, ByVal iModeNum As Long, lpDevMode As DEVMODE) As Long

Public Const ENUM_CURRENT_SETTINGS = -1
Public Const ENUM_REGISTRY_SETTINGS = -2

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim dm As DEVMODE ' info about the display device
    Dim retval As Long ' return value

    ' Initialize the structure.
    dm.dmSize = Len(dm)
    ' Get the display settings for the current monitor and mode.
    retval = EnumDisplaySettings(vbNullString, ENUM_CURRENT_SETTINGS, dm)
    ' Print some information about the display.
    Debug.Print "- Bits Per Pixel: "; dm.dmBitsPerPixel
    Debug.Print "- Width (Pixels): "; dm.dmPelsWidth
    Debug.Print "- Height (Pixels): "; dm.dmPelsHeight
    Debug.Print "- Display Freq.: "; dm.dmDisplayFrequency

End Sub

See Also

ChangeDisplaySettings

Category
## EnumFontFamilies Function

Declare Function EnumFontFamilies Lib "gdi32.dll" Alias "EnumFontFamiliesA" (ByVal hdc As Long, ByVal lpszFamily As Any, ByVal lpEnumFontFamProc As Long, ByVal lParam As Long) As Long

### Platforms

- **Windows 95**: Supported but Obsolete; use EnumFontFamiliesEx instead.
- **Windows 98**: Supported but Obsolete; use EnumFontFamiliesEx instead.
- **Windows NT**: Requires Windows NT 3.1 or later but Obsolete under Windows NT 4.0 or later; use EnumFontFamiliesEx instead.
- **Windows 2000**: Supported but Obsolete; use EnumFontFamiliesEx instead.
- **Windows CE**: Not Supported.

### Description & Usage

*EnumFontFamilies* enumerates all of the fonts available for use on a *device* which use a certain typeface. The only trait that the function looks for in the enumerated fonts is that it uses the specified typeface. The enumerated fonts are individually passed to a *callback function* for processing.

### Return Value

The function returns whatever the final call to the callback function returned.

### Visual Basic-Specific Issues

When passing 0 as *lpszFamily*, the expression CLng(0) must be used.

### Parameters

- **hdc**
  A *handle* to a *device context* to the device to enumerate the fonts of.
- **lpszFamily**
  The name of the font typeface which the enumerated fonts must use. To instead enumerate a single font from each possible typeface, pass 0 for this parameter (not an empty string!).
- **lpEnumFontFamProc**
  A *pointer* to the *EnumFontFamProc* callback function which processes the information about each font that is enumerated.
Windows API Guide: EnumFontFamilies Function

 lParam
 A value to pass to the function specified by lpEnumFontFamProc.

Example

' This code is licensed according to the terms and conditions listed here.

' Enumerate some of the fonts available for window Form1.
' These fonts must have the Times New Roman typeface.
' Display some information about each font as it is enumerated.

' *** Place the following code in a module. ***
' The following callback function processes the enumerated fonts.
Public Function EnumFontFamProc (ByVal lpelf As Long, ByVal lpntm As Long, ByVal
FontType As Long, ByVal lParam As Long) As Long
Dim elf As ENUMLOGFONT ' receives information about the font
Dim ntm As NEWTEXTMETRIC ' receives text metrics for TrueType fonts
Dim tm As TEXTMETRIC ' receives text metrics for non-TrueType fonts

' Copy the font information into the appropriate structure.
CopyMemory elf, ByVal lpelf, Len(elf)

' If the font is TrueType, use the following code.
If (FontType And TRUETYPE_FONTTYPE) = TRUETYPE_FONTTYPE Then
  CopyMemory ntm, ByVal lpntm, Len(ntm)
  ' Display the name of the font (removing empty space from it).
  Debug.Print "Font Name: "; Left(elf.elfFullName, InStr(elf.elfFullName, vbNullChar) - 1);
  Debug.Print " (TrueType font)"
  ' Display the style of the font (again removing empty space).
  Debug.Print "Font Style: "; Left(elf.elfStyle, InStr(elf.elfStyle, vbNullChar) - 1)
  ' Display the average character width.
  Debug.Print "Average Character Width:"; ntm.tmAveCharWidth
  ' Display the maximum character width.
  Debug.Print "Maximum Character Width:"; ntm.tmMaxCharWidth

' If the font is not TrueType, use the following code.
Else
  CopyMemory tm, ByVal lpntm, Len(tm)
  ' Display the name of the font (removing empty space from it).
  Debug.Print "Font Name: ";
  Debug.Print Left(elf.elfLogFont.lfFaceName, InStr(elf.elfLogFont.lfFaceName, vbNullChar) - 1);
  ' Display whether the font is a device or a raster font.
  If FontType = DEVICE_FONTTYPE Then
    Debug.Print " (Device font)"
  ElseIf FontType = RASTER_FONTTYPE Then
    Debug.Print " (Raster font)"

End If
Debug.Print "Font Style does not apply for this font."
' Display the average character width.
Debug.Print "Average Character Width:"; tm.tmAveCharWidth
' Display the maximum character width.
Debug.Print "Maximum Character Width:"; tm.tmMaxCharWidth
End If

Debug.Print "***" ' separator
' Tell EnumFontFamilies to continue enumeration.
EnumFontFamProc = 1
End Function

' *** Place this code wherever you want the enumerate the fonts. ***
Dim retval As Long ' return value

' Enumerate all the fonts with the Times New Roman ' typeface which are available on Form1.
retval = EnumFontFamilies(Form1.hDC, "Times New Roman", AddressOf EnumFontFamProc, 0)
Debug.Print "Enumeration complete."

See Also

EnumFontFamiliesEx

Category

Fonts & Text

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 29, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/enumfontfamilies.html
EnumFontFamiliesEx Function

Declare Function EnumFontFamiliesEx Lib "gdi32.dll" Alias "EnumFontFamiliesExA" (ByVal hdc As Long, lpLogfont As LOGFONT, ByVal lpEnumFontFamExProc As Long, ByVal lParam As Long, ByVal dwFlags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

EnumFontFamiliesEx enumerates all of the fonts available for use on a device which match a basic description. The function enumerates fonts based on their character sets and/or typeface styles (or simply all fonts). The function treats identical fonts with different character sets as two different fonts (for example, Times New Roman with the ANSI character set is considered a different font than Times New Roman with the Cyrillic character set). The fonts are not enumerated in any obvious order.

Return Value

The function returns whatever the final call to the callback function returned.

Visual Basic-Specific Issues

None.

Parameters

- **(hdc)**
  
  A handle to a device context to the device to enumerate the available fonts of.

- **lpLogfont**
  
  Identifies the character set and/or typeface name (or neither) of the fonts to enumerate. The .lfCharset member identifies the character set of the fonts to enumerate, or DEFAULT_CHARSET to ignore the character set. The .lfFaceName member identifies the typeface name of the fonts to enumerate, or an empty string to ignore the typeface name. The .lfPitchAndFamily member is not used but must be zero. All other members are ignored.

- **lpEnumFontFamExProc**
  
  A pointer to the EnumFontFamExProc callback function which processes the information about each font that is...
Windows API Guide: EnumFontFamiliesEx Function

enumerator.

lParam
A value to pass to the function identified by lpEnumFontFamExProc.

dwFlags
Reserved -- set to 0.

Example

' This code is licensed according to the terms and conditions listed here.

' Enumerate some of the fonts available for window Form1.
' These fonts must have the ANSI character set and have the
' Times New Roman typeface. Display some information about
' each font as it is enumerated.

' *** Place the following code in a module. ***
' The following callback function processes the enumerated fonts.
Public Function EnumFontFamExProc (ByVal lpelfe As Long, ByVal lpntme As Long, ByVal
FontType As Long, ByVal lParam As Long) As Long
Dim elfx As ENUMLOGFONTEX ' receives information about the font
Dim ntmx As NEWTEXTMETRICEX ' receives text metrics for TrueType fonts
Dim tm As TEXTMETRIC ' receives text metrics for non-TrueType fonts

' Copy the font information into the appropriate structure.
CopyMemory elfx, ByVal lpelfe, Len(elfx)

' If the font is TrueType, use the following code.
If (FontType And TRUETYPE_FONTTYPE) = TRUETYPE_FONTTYPE Then
  ' Copy the text metrics into the appropriate structure.
  CopyMemory ntmx, ByVal lpntme, Len(ntmx)
  ' Display the name of the font (removing empty space from it).
  Debug.Print "Font Name: "; Left(elfx.elfFullName, InStr(elfx.elfFullName, vbNullChar) - 1);
  Debug.Print " (TrueType font)"
  ' Display the style of the font (again removing empty space).
  Debug.Print "Font Style: "; Left(elfx.elfStyle, InStr(elfx.elfStyle, vbNullChar) - 1)
  ' Display the average character width.
  Debug.Print "Average Character Width:"; ntmx.ntmTm.tmAveCharWidth
  ' Display the maximum character width.
  Debug.Print "Maximum Character Width:"; ntmx.ntmTm.tmMaxCharWidth
Else
  ' Copy the text metrics into the appropriate structure.
  CopyMemory tm, ByVal lpntme, Len(tm)
  ' Display the name of the font (removing empty space from it).
  Debug.Print "Font Name: "; Left(elfx.elfLogFont.lfFaceName, InStr(elfx.elfLogFont.lfFaceName, vbNullChar) - 1);
  ' Display whether the font is a device or a raster font.
  If FontType = DEVICE_FONTTYPE Then
Debug.Print " (Device font)"
ElseIf FontType = RASTER_FONTTYPE Then
  Debug.Print " (Raster font)"
End If

Debug.Print "Font Style does not apply for this font."
' Display the average character width.
Debug.Print "Average Character Width:"; tm.tmAveCharWidth
' Display the maximum character width.
Debug.Print "Maximum Character Width:"; tm.tmMaxCharWidth
End If

Debug.Print "***" ' separator
' Tell EnumFontFamiliesEx to continue enumeration.
EnumFontFamExProc = 1
End Function

' *** Place this code wherever you want the enumerate the fonts. ***
Dim lf As LOGFONT ' describes enumeration attributes
Dim retval As Long ' return value

' Initialize the structure to describe the fonts we want.
lf.lfCharset = ANSI_CHARSET ' fonts with the ANSI character set
lf.lfFaceName = "Times New Roman" & vbNullChar ' fonts with the Times New Roman typeface
lf.lfPitchAndFamily = 0 ' this must be 0
' Enumerate such fonts available on window Form1.
retval = EnumFontFamiliesEx(Form1.hDC, lf, AddressOf EnumFontFamExProc, 0, 0)
Debug.Print "Enumeration complete."

See Also

EnumFontFamilies

Category

Fonts & Text

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 23, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/enumfontfamiliesex.html
EnumJobs Function

Declare Function EnumJobs Lib "winspool.drv" Alias "EnumJobsA" (ByVal hPrinter As Long, ByVal FirstJob As Long, ByVal NoJobs As Long, ByVal Level As Long, pJob As Long, ByVal cbBuf As Long, pcbNeeded As Long, pcReturned As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

EnumJobs enumerates all of the print jobs pending for a given printer. The function can retrieve either (relatively) brief or detailed information describing each job in the print queue for that printer. Note that, instead of placing the information directly into the corresponding structures, the function places the information into an array. The array's contents must then be transferred to one or more structures. In order to determine how much space is necessary to receive the information about each print job, first call the function with cbBuf set to 0. The function will set the variable passed as pcbNeeded to the size in bytes of the array necessary hold the information.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

The data retrieved by the function cannot be directly copied into structure(s) using CopyMemory or some other method. Instead, each (desired) member of the structure must manually be copied from the array to the structure, using the corresponding array indices. Each structure's contents appear sequentially in the array, and one structure immediately follows another. See the example code for a demonstration on how this is done.

Additionally, strings are not stored directly. Instead, the data placed into the array contains pointers to strings whenever necessary. The string manipulation API functions lstrlen and lstrcpy are needed to copy the pointed-to string into the structure's string data member(s). Again, see the example for a clearer demonstration. However, to copy structures stored under the original structure, CopyMemory can be used. Of course, numeric data members can be copied directly from the array to the structure.
Finally, when determining the space needed for the array, both \textit{pJob} and \textit{cbBuf} must be set to 0. To set \textit{pJob} to 0, the expression \texttt{ByVal 0} must be used.

\section*{Parameters}

\textbf{hPrinter}

A \texttt{handle} to the printer to enumerate the print jobs of.

\textbf{FirstJob}

The zero-baded position in the print queue to begin print job enumeration from. To enumerate all jobs, of course set this to 0. To enumerate everything except the first job, set this to 1 (identifying the second print job), etc.

\textbf{NoJobs}

The maximum number of print jobs to enumerate.

\textbf{Level}

Identifies the format of the information placed into the array passed as \textit{pJob}. If this is 1, the data is in the format of \texttt{JOB\_INFO\_1} structures. If this is 2, the data is in the format of \texttt{JOB\_INFO\_2} structures.

\textbf{pJob}

An array which receives all the information about the print jobs retrieved by the function. The format of this data is specified in \textit{Level}. The array must be large enough to receive all the data. If you are setting \textit{cbBuf} to 0, this must also be 0.

\textbf{cbBuf}

The size in bytes of the array passed as \textit{pJob}. To determine how large the array must be, set this to 0.

\textbf{pcbNeeded}

Receives the number of bytes of data copied into the array passed as \textit{pJob}. If the function failed, this is the minimum size in bytes which the array passed as \textit{pJob} must be.

\textbf{pcReturned}

Receives the number of print jobs retrieved by the function.

\section*{Example}

\begin{verbatim}
' This code is licensed according to the terms and conditions listed here.

' Display a little information about each print job in the
' queue for the default printer. Note how both EnumPrinters and
' EnumJobs first copy the data into the arrays before placing
' the information into the desired structure.
Dim pil As PRINTER\_INFO\_1  ' holds a little information about the printer
Dim hPrinter As Long  ' handle to the default printer once it is opened
Dim arraybuf() As Long  ' resizable array used as a buffer
Dim jobinfo As JOB\_INFO\_2  ' holds detailed info about a print job
Dim needed As Long  ' receives space needed in the buffer array
Dim numitems As Long  ' receives the number of items returned
Dim lendivfour As Long  ' the size in Long-type units of the jobinfo structure
Dim c As Long  ' counter variable
Dim retval As Long  ' return value

' -- Get the name of the default printer. --
' Determine how much space is needed to get the printer information.
retval = EnumPrinters(PRINTER\_ENUM\_DEFAULT, ",", 1, ByVal 0, 0, needed, numitems)
' Resize the array buffer to the needed size in bytes.

\end{verbatim}
ReDim arraybuf(0 To needed / 4 - 1)  ' remember each element is 4 bytes
' Retrieve the information about the default printer.
retval = EnumPrinters(PRINTER_ENUM_DEFAULT, "", 1, arraybuf(0), needed, needed, numitems)
' Copy the printer name into the structure. The rest is unnecessary.
pil.pName = Space(lstrlen(arraybuf(2)))
retval = lstrcpy(pil.pName, arraybuf(2))

' -- Obtain a handle to the default printer (using default configuration). --
retval = OpenPrinter(pil.pName, hPrinter, ByVal CLng(0))

' -- Enumerate the default printer's print jobs currently queued. --
' Determine how much space is needed to get the print jobs' information.
retval = EnumJobs(hPrinter, 0, 100, 2, ByVal 0, 0, needed, numitems)
' Resize the array buffer to the needed size in bytes.
ReDim arraybuf(0 To needed / 4 - 1)  ' remember each element is 4 bytes
' Retrieve the information about the print jobs.
retval = EnumJobs(hPrinter, 0, 100, 2, arraybuf(0), needed, needed, numitems)
' Display the number of print jobs currently in the queue.
If numitems > 0 Then
  Debug.Print "There are"; numitems; "print jobs currently in the queue."
Else
  Debug.Print "No print jobs are currently in the queue."
End If

' For each print job, copy its data into the structure. Then display selected
' information from the structure. For brevity, this example copies only a
' few of the data members into the structure.
lendivfour = Len(jobinfo) / 4  ' this is the number of elements for each structure in
the array
For c = 0 To numitems - 1  ' loop through each item
  ' Copy selected information into the structure: the job ID number, the
  ' name of the user who printed it, the total number of pages, and
  ' the time it was added into the queue.
  jobinfo.JobID = arraybuf(lendivfour * c)  ' the first element of the array chunk
  jobinfo.pUserName = Space(lstrlen(arraybuf(lendivfour * c + 3)))  ' fourth element
  retval = lstrcpy(jobinfo.pUserName, arraybuf(lendivfour * c + 3))
  jobinfo.TotalPages = arraybuf(lendivfour * c + 18)  ' nineteenth element
  CopyMemory jobinfo.Submitted, arraybuf(lendivfour * c + 20), Len(jobinfo.Submitted)
  ' twenty-first element

  ' Display the copied information.
  Debug.Print "Job ID number:"; jobinfo.JobID
  Debug.Print "Printed by user: "; jobinfo.pUserName
  Debug.Print "Number of pages:"; jobinfo.TotalPages
  Debug.Print "Placed in queue on: ";
  ' (display the date and time stored in jobinfo.Submitted)
  Debug.Print jobinfo.Submitted.wMonth; "-"; jobinfo.Submitted.wDay; "-";
  jobinfo.Submitted.wYear; " ";
  Debug.Print jobinfo.Submitted.wHour; ":"; jobinfo.Submitted.wMinute; ":";
  jobinfo.Submitted.wSecond; " GMT"
Next c
' Close the printer handle now that it is no longer needed.
retval = ClosePrinter(hPrinter)

Category

Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.
EnumPrinters Function

Declare Function EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal flags As Long, ByVal name As String, ByVal Level As Long, pPrinterEnum As Long, ByVal cdBuf As Long, pcbNeeded As Long, pcReturned As Long) As Long

Platforms: Win 95/98, Win NT

EnumPrinters finds and returns information about one or more printers which the computer has access to. These include both local printers (physically connected to the machine) and network printers (accessible via the network). Under Win 95/98, the information can be passed in a PRINTER_INFO_1, PRINTER_INFO_2, or PRINTER_INFO_5 structure. Under Win NT, the information can be passed in a PRINTER_INFO_1, PRINTER_INFO_2, or PRINTER_INFO_4 structure. Note that structures 4 and 5 are the quickest ones to use. The attributes of the chosen structure determine what kinds of information about the printer(s) is returned. The information itself is put into the array passed as pPrinterEnum, which can then be copied into an array of data structures. The function returns 1 if successful, or 0 if an error occurred.

Note for Visual Basic users: Due to limitations in the Visual Basic language, it is impossible to pass an array of PRINTER_INFO_* structures as pPrinterEnum -- the array must be of Long-type elements. Also, for some reason the CopyMemory function cannot be used to successfully transfer the array information into the structure. Instead, each individual member of the data structure must be set manually. Look at the two examples for details how. In order to convert a Long-type string pointer (in the array) to a string, variants of the lstrcpy and lstrlen function must be used. Again, see the examples. CopyMemory, however, can be used to copy the data structures which make up parts of the PRINTER_INFO_2 data structure.

flags
One or more of the following flags specifying which printers to find information about (note that PRINTER_ENUM_LOCAL and PRINTER_ENUM_CONNECTIONS are the only valid flags when using PRINTER_INFO_4):

PRINTER_ENUM_CONNECTIONS = &H4  
Win NT only: Get information about the network printers which the computer has made connections to.

PRINTER_ENUM_DEFAULT = &H1  
Win 95/98 only: Get information about the computer's default printer.

PRINTER_ENUM_LOCAL = &H2  
Get information about local printers (the ones directly connected to the system). Win 95/98 for some reason also considers network printers to be local.

PRINTER_ENUM_NAME = &H8  
Get information about all the printers under the network domain specified by name.

PRINTER_ENUM_NETWORK = &H40  
Win NT only: Get information about all the printers under the computer's domain in the network. This only works with the PRINTER_INFO_1 structure.

PRINTER_ENUM_REMOTE = &H10  
Win NT only: Same as PRINTER_ENUM_NETWORK.

PRINTER_ENUM_SHARED = &H20  
Get information about all the printers with the shared attribute.

name
Windows API Guide: EnumPrinters Function

The name of the network domain to look under, if applicable. See flags to see when this would be used. If not used, set this parameter to an empty string.

Level
Specifies which PRINTER_INFO_* structure to use. For Win 95/98, this can be 1, 2, or 5. For Win NT, this can be 1, 2, or 4.

pPrinterEnum
An array which receives all of the information found by the function. This needs to be copied manually into the PRINTER_INFO_* structure.

cBuf
The size in bytes of the array passed as pPrinterEnum.

pcbNeeded
If successful, receives the number of bytes of information the function found. If unsuccessful, receives the number of bytes that pPrinterEnum must have in order to receive all of the information.

pcReturned
Receives the number of printers found by the function.

Example #1:

' Get information about all of the local printers using structure 1. Note how the elements of the array are loaded into an array of data structures manually. Also note how the following special declares must be used to allow numeric string pointers to be used in place of strings:
Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

Dim longbuffer() As Long  ' resizable array receives information from the function
Dim printinfo() As PRINTER_INFO_1  ' values inside longbuffer() will be put into here
Dim numbytes As Long  ' size in bytes of longbuffer()
Dim numneeded As Long  ' receives number of bytes necessary if longbuffer() is too small
Dim numprinters As Long  ' receives number of printers found
Dim c As Integer, retval As Long  ' counter variable & return value

' Get information about the local printers
numbytes = 3076  ' should be sufficiently big, but it may not be
ReDim longbuffer(0 To numbytes / 4) As Long  ' resize array -- note how 1 Long = 4 bytes
retval = EnumPrinters(PRINTER_ENUM_LOCAL, ",", 1, longbuffer(0), numbytes, numneeded, numprinters)
If retval = 0 Then  ' try enlarging longbuffer() to receive all necessary information
    numbytes = numneeded
    ReDim longbuffer(0 To numbytes / 4) As Long  ' make it large enough
    retval = EnumPrinters(PRINTER_ENUM_LOCAL, ",", 1, longbuffer(0), numbytes, numneeded, numprinters)
    If retval = 0  ' failed again!
        Debug.Print "Could not successfully enumerate the printes."
        End ' abort program
    End If
End If

' Convert longbuffer() data into printinfo()
ReDim printinfo(0 To numprinters - 1) As PRINTER_INFO_1  ' room for each printer
For c = 0 To numprinters - 1  ' loop, putting each set of information into each element
    ' longbuffer(4 * c) = .flags, longbuffer(4 * c + 1) = .pDescription, etc.
    ' For each string, the string is first buffered to provide enough room, and then the string is copied.
    printinfo(c).flags = longbuffer(4 * c)
    printinfo(c).pDescription = Space(lstrlen(longbuffer(4 * c + 1)))
    retval = lstrcpy(printinfo(c).pDescription, longbuffer(4 * c + 1))
    printinfo(c).pName = Space(lstrlen(longbuffer(4 * c + 2)))
    retval = lstrcpy(printinfo(c).pName, longbuffer(4 * c + 2))
    printinfo(c).pComment = Space(lstrlen(longbuffer(4 * c + 3)))
    retval = lstrcpy(printinfo(c).pComment, longbuffer(4 * c + 3))
Next c

' Display name of each printer
For c = 0 To numprinters - 1
    Debug.Print "Name of printer"; c + 1; " is: "; printinfo(c).pName
Next c

Example #2:

' Get information about all of the local printers using structure 2. Note how the elements of the array are loaded into an array of data structures manually. Also note how the following special declares must be used to allow numeric string pointers to be used in place of strings:
Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

Dim longbuffer() As Long  ' resizable array receives information from the function
Dim printinfo() As PRINTER_INFO_2  ' values inside longbuffer() will be put into here
Dim numbytes As Long  ' size in bytes of longbuffer()
Dim numneeded As Long  ' receives number of bytes necessary if longbuffer() is too small
Dim numprinters As Long  ' receives number of printers found
Dim c As Integer, retval As Long  ' counter variable & return value

' Get information about the local printers
numbytes = 3076  ' should be sufficiently big, but it may not be
ReDim longbuffer(0 To numbytes / 4) As Long  ' resize array -- note how 1 Long = 4 bytes
retval = EnumPrinters(PRINTER_ENUM_LOCAL, ",", 2, longbuffer(0), numbytes, numneeded, numprinters)
If retval = 0 Then  ' try enlarging longbuffer() to receive all necessary information
numbytes = numneeded
ReDim longbuffer(0 To numbytes / 4) As Long ' make it large enough
retval = EnumPrinters(PRINTER_ENUM_LOCAL, "", 2, longbuffer(0), numbytes, numneeded, numprinters)
If retval = 0 ' failed again!
    Debug.Print "Could not successfully enumerate the printes."
    End ' abort program
End If

' Convert longbuffer() data into printinfo()
ReDim printinfo(0 To numprinters - 1) As PRINTER_INFO_2 ' room for each printer
For c = 0 To numprinters - 1 ' loop, putting each set of information into each element
    longbuffer(21 * c) = .pServerName, longbuffer(21 * c + 1) = .pPrinterName, etc.
    ' For each string, the string is first buffered to provide enough room, and then
    ' the string is copied.
    ' For each structure, the memory of it is directly copied via CopyMemory using the pointer.
    printinfo(c).pServerName = Space(lstrlen(longbuffer(21 * c)))
    retval = lstrcpy(printinfo(c).pServerName, longbuffer(21 * c))
    printinfo(c).pPrinterName = Space(lstrlen(longbuffer(21 * c + 1)))
    retval = lstrcpy(printinfo(c).pPrinterName, longbuffer(21 * c + 1))
    printinfo(c).pShareName = Space(lstrlen(longbuffer(21 * c + 2)))
    retval = lstrcpy(printinfo(c).pShareName, longbuffer(21 * c + 2))
    printinfo(c).pPortName = Space(lstrlen(longbuffer(21 * c + 3)))
    retval = lstrcpy(printinfo(c).pPortName, longbuffer(21 * c + 3))
    printinfo(c).pDriverName = Space(lstrlen(longbuffer(21 * c + 4)))
    retval = lstrcpy(printinfo(c).pDriverName, longbuffer(21 * c + 4))
    printinfo(c).pComment = Space(lstrlen(longbuffer(21 * c + 5)))
    retval = lstrcpy(printinfo(c).pComment, longbuffer(21 * c + 5))
    printinfo(c).pLocation = Space(lstrlen(longbuffer(21 * c + 6)))
    retval = lstrcpy(printinfo(c).pLocation, longbuffer(21 * c + 6))
    CopyMemory printinfo(c).pDevMode, longbuffer(21 * c + 7), Len(printinfo(c).pDevMode)
    printinfo(c).pSepFile = Space(lstrlen(longbuffer(21 * c + 8)))
    retval = lstrcpy(printinfo(c).pSepFile, longbuffer(21 * c + 8))
    printinfo(c).pPrintProcessor = Space(lstrlen(longbuffer(21 * c + 9)))
    retval = lstrcpy(printinfo(c).pPrintProcessor, longbuffer(21 * c + 9))
    printinfo(c).pDatatype = Space(lstrlen(longbuffer(21 * c + 10)))
    retval = lstrcpy(printinfo(c).pDatatype, longbuffer(21 * c + 10))
    printinfo(c).pParameters = Space(lstrlen(longbuffer(21 * c + 11)))
    retval = lstrcpy(printinfo(c).pParameters, longbuffer(21 * c + 11))
    CopyMemory printinfo(c).pSecurityDescriptor, longbuffer(21 * c + 12),
    Len(printinfo(c).pSecurityDescriptor)
    printinfo(c).Attributes = longbuffer(21 * c + 13)
    printinfo(c).Priority = longbuffer(21 * c + 14)
    printinfo(c).DefaultPriority = longbuffer(21 * c + 15)
    printinfo(c).StartTime = longbuffer(21 * c + 16)
printinfo(c).UntilTime = longbuffer(21 * c + 17)
printinfo(c).Status = longbuffer(21 * c + 18)
printinfo(c).cJobs = longbuffer(21 * c + 19)
printinfo(c).AveragePPM = longbuffer(21 * c + 20)
Next c

' Display the name of each printer and its average page per minute (ppm) rate
For c = 0 To numprinters - 1
    Debug.Print printinfo(c).pPrinterName; " prints an average of";
    printinfo(c).AveragePPM; "pages per minute."
Next c

Category: Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.
EnumPropsEx Function

Declare Function EnumPropsEx Lib "user32.dll" Alias "EnumPropsExA" (ByVal hWnd As Long, ByVal lpEnumFunc As Long, ByVal lParam As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

EnumPropsEx enumerates all of the *window properties* of a window. Information about each property is passed, one at a time, to the specified *callback function*. The window properties are not enumerated in any particular order.

Return Value

If the window had no window properties set, the function returns -1. Otherwise, the function returns whatever the most recent value the callback function returned was.

Visual Basic-Specific Issues

None.

Parameters

- **hWnd**
  A handle to the window to enumerate the window properties of.
- **lpEnumFunc**
  A pointer to the PropEnumProcEx callback function which receives information about each enumerated window property.
- **lParam**
  An additional value to pass to the function specified by *lpEnumFunc*.
Example

' This code is licensed according to the terms and conditions listed here.

' List the names of each window property belonging to window Form1. This example does not display the actual data in the properties because the callback function is not told what the handle for each property refers to.

' *** Place the following code in a module. ***
' This callback function merely displays the name of the window property it is given.
Public Function PropEnumProcEx(ByVal hwnd As Long, ByVal lpszString As Long, ByVal hData As Long, ByVal dwData As Long) As Long
    Dim propname As String  ' receives the name of the window property
    Dim retval As Long  ' generic return value
    ' Copy the string pointed to by lpString into a "real" string.
    propname = Space(lstrlen(lpszString))
    retval = lstrcpy(propname, lpszString)
    ' Display the property name (not including its value).
    Debug.Print "- "; propname
    ' Tell EnumPropsEx to continue enumeration.
    PropEnumProcEx = 1
End Function

' *** Place the following code wherever you wish to enumerate the properties.
Dim retval As Long  ' result of enumeration
Debug.Print "BEGINNING ENUMERATION OF Form1's PROPERTIES"
' Enumerate the properties of window Form1.
retval = EnumPropsEx(Form1.hWnd, AddressOf PropEnumProcEx, 0)
If retval = -1 Then  ' no properties to enumerate
    Debug.Print "(no window properties were found)"
End If

Category

Window Properties

Go back to the alphabetical Function listing. Go back to the Reference section index.

---

Last Modified: December 24, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
EnumThreadWindows Function

Declare Function EnumThreadWindows Lib "user32.dll" (ByVal dwThreadId As Long, ByVal lpfn As Long, ByVal lParam As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

EnumThreadWindows enumerates and provides handles to all of the windows owned and controlled by a given thread. (Note that these windows include many windows not visible to the user.) Each time a window is located, the function passes that handle to an application-defined callback function. The function continues doing so until all windows have been enumerated, or until the process has been aborted.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- dwThreadId
  An identifier to the thread to enumerate the windows of.
- lpfn
  A pointer to the application-defined callback function EnumThreadWndProc.
- lParam
  An additional value to pass to the application-defined callback function.

Example
This code is licensed according to the terms and conditions listed here.

Display the window title of all windows controlled by the thread which the window Form1 is in. This task is given to the callback function, which will receive each handle individually. Note that if the window has no title bar text, it will not be displayed (for clarity's sake).

*** Place this code in a module. This is the callback function. ***
Public Function EnumThreadWndProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
Dim slength As Long, wintext As String ' title bar text length and buffer
Dim retval As Long ' return value
Static winnum As Integer ' counter keeps track of how many windows have been enumerated

winnum = winnum + 1 ' one more window enumerated....
slength = GetWindowTextLength(hwnd) + 1 ' get length of title bar text
If slength > 1 ' if return value refers to non-empty string
    buffer = Space(slength) ' make room in the buffer
    retval = GetWindowText(hwnd, buffer, slength) ' get title bar text
    Debug.Print "Window "; winnum; ": "; ' display number of enumerated window
    Debug.Print Left(buffer, slength - 1) ' display title bar text of enumerated window
End If

EnumThreadWndProc = 1 ' return value of 1 means continue enumeration
End Function

*** Place this code wherever you want to enumerate the windows. ***
Dim threadid As Long, processid As Long ' receive id to thread and process of Form1
Dim retval As Long ' return value

' Determine the thread which owns the window Form1.
threadid = GetWindowThreadProcessId(Form1.hWnd, processid)
' Use the callback function to list all of the enumerated thread windows. Note that lParam
' is set to 0 because we don't need to pass any additional information to the function.
retval = EnumThreadWindows(threadid, AddressOf EnumThreadWndProc, 0)

See Also
EnumChildWindows, EnumWindows

Category
Windows
EnumWindows Function

Declare Function EnumWindows Lib "user32.dll" (ByVal lpEnumFunc As Long, ByVal lParam As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

EnumWindows enumerates and provides handles to all of the currently open top-level windows. This function will ignore child windows. (Note that the top-level windows include many windows not visible to the user.) Each time a window is located, the function passes that handle to an application-defined callback function. The function continues doing so until all windows have been enumerated, or until the process has been aborted.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpEnumFunc

A pointer to the application-defined callback function EnumWindowsProc.

lpParam

An additional value to pass to the application-defined callback function.

Example:

' Display the title bar text of all top-level windows. This
' task is given to the callback function, which will receive each handle individually.
' Note that if the window has no title bar text, it will not be displayed (for clarity's sake).

' *** Place this code in a module. This is the callback function. ***
Public Function EnumWindowsProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
    Dim slength As Long, buffer As String  ' title bar text length and buffer
    Dim retval As Long  ' return value
    Static winnum As Integer  ' counter keeps track of how many windows have been enumerated

    winnum = winnum + 1  ' one more window enumerated....
    slength = GetWindowTextLength(hwnd) + 1  ' get length of title bar text
    If slength > 1  ' if return value refers to non-empty string
        buffer = Space(slength)  ' make room in the buffer
        retval = GetWindowText(hwnd, buffer, slength)  ' get title bar text
        Debug.Print "Window "; winnum; ": ";  ' display number of enumerated window
        Debug.Print Left(buffer, slength - 1)  ' display title bar text of enumerated window
    End If

    EnumWindowsProc = 1  ' return value of 1 means continue enumeration
End Function

' *** Place this code wherever you want to enumerate the windows. ***
Dim retval As Long  ' return value

' Use the above callback function to list all of the enumerated windows. Note that lParam is
' set to 0 because we don't need to pass any additional information to the function.
retval = EnumWindows (AddressOf EnumWindowsProc, 0)

See Also

EnumChildWindows, EnumThreadWindows

Category

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: January 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
EqualRect Function

Declare Function EqualRect Lib "user32.dll" (lpRect1 As RECT, lpRect2 As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

EqualRect determines if two rectangles are equal. Rectangles are considered equal if and only if the upper-left and lower-right corners (the points that define the rectangles) of one rectangle are equal to those of another. The function returns 1 if the two rectangles are equal and 0 if they are unequal.

lpRect1
The first of the two rectangles to check.

lpRect2
The second of the two rectangles to check.

Example:

' Demonstrate equal and unequal rectangles
Dim r As RECT, s As RECT  ' rectangles to use
Dim areequal As Long  ' receives whether the rectangles are equal or not
Dim retval As Long  ' return value

' Initialize the two rectangles using the API
retval = SetRect(r, 15, 20, 100, 110)  ' r = (15,20)-(100,110)
retval = SetRect(s, 15, 20, 100, 110)  ' s = (15,20)-(100,110)
areequal = EqualRect(r, s)  ' compare the rectangles
If areequal = 1 Then Debug.Print "Are Equal" Else Debug.Print "Are Not Equal"

' Change the second rectangle
retval = SetRect(s, 30, 45, 200, 250)  ' s = (30,45)-(200,250)
areequal = EqualRect(r, s)  ' compare the rectangles
If areequal = 1 Then Debug.Print "Are Equal" Else Debug.Print "Are Not Equal"
' The first time is Are Equal, the second is Are Not Equal.

See Also: CopyRect
Category: Rectangles

Go back to the alphabetical Function listing.
EqualRgn Function

Declare Function EqualRgn Lib "gdi32.dll" (ByVal hSrcRgn1 As Long, ByVal hSrcRgn2 As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**EqualRgn** determines if two regions contain the exact same area. Although the region **handles** will of course be different, they could still refer to regions of identical size, shape, and position.

Return Value

If the two regions are equal, the function returns a non-zero value. If the two regions are different, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

- **hSrcRgn1**
  
  A handle to the first of the two regions to compare.

- **hSrcRgn2**
  
  A handle to the second of the two regions to compare.

Example

Perform a simple illustration of equal and unequal regions. Besides showing how the function works, there's no practical value to this example. The example runs when you click on command button Command1 in a form window. So, naturally, there has to be a command button named Command1 on that window.
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/e/equalrgn.html).

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

```vba
Public Declare Function EqualRgn Lib "gdi32.dll" (ByVal hSrcRgn1 As Long, ByVal hSrcRgn2 As Long) As Long
Public Declare Function CreateEllipticRgn Lib "gdi32.dll" (ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long) As Long
Public Declare Function DeleteObject Lib "gdi32.dll" (ByVal hObject As Long) As Long
```

```vba
' *** Place the following code inside a form. ***

Private Sub Command1_Click()
    Dim hRgn1 As Long, hRgn2 As Long, hRgn3 As Long ' the three regions
    Dim areequal As Long ' receives equal/unequal indicator
    Dim retval As Long ' generic return value

    ' Define all three regions as elliptical.
    hRgn1 = CreateEllipticRgn(20, 30, 120, 80)
    hRgn2 = CreateEllipticRgn(20, 30, 120, 80)
    hRgn3 = CreateEllipticRgn(50, 50, 200, 150)

    ' Compare regions 1 and 2 (they will be equal).
    areequal = EqualRgn(hRgn1, hRgn2)
    If areequal = 0 Then Debug.Print "Not Equal" Else Debug.Print "Equal"

    ' Compare regions 1 and 3 (they will not be equal).
    areequal = EqualRgn(hRgn1, hRgn3)
    If areequal = 0 Then Debug.Print "Not Equal" Else Debug.Print "Equal"

    ' Delete the three regions to free up resources.
    retval = DeleteObject(hRgn1)
    retval = DeleteObject(hRgn2)
    retval = DeleteObject(hRgn3)
End Sub
```

**Category**

Regions

ExitWindowsDialog Function

Declare Sub ExitWindowsDialog Lib "shell32.dll" Alias "#60" (ByVal hwndOwner As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Unknown.

The ExitWindowsDialog function is officially undocumented.

Description & Usage

ExitWindowsDialog prompts the user with the "Shut Down Windows" dialog box. This is the same box as the one that appears when the user selects "Shut Down" from the Start menu. Unfortunately, there is no way to know what the user chose in the dialog box, or if the user simply cancelled it. Unlike most API dialogs, this function actually begins the shut down, reboot, or whatever other process the user selected.

Return Value

This function does not return a value.

Visual Basic-Specific Issues

None.

Parameters

hwndOwner

A handle to the window that is opening the Shut Down dialog box.

Example

' This code is licensed according to the terms and conditions listed here.
Public Declare Sub ExitWindowsDialog Lib "shell32.dll" Alias "#60" (ByVal hwndOwner As Long)

Private Sub Command1_Click()
    ExitWindowsDialog Form1 hWnd
    ' That's all it takes!
End Sub

See Also

RestartDialog

Category

Shell

Go back to the Reference section.
ExitWindowsEx Function

Declare Function ExitWindowsEx Lib "user32.dll" (ByVal uFlags As Long, ByVal dwReserved As Long) As Long

Platforms: Win 95/98, Win NT

ExitWindowsEx shuts down or reboots the user's computer. Of course, since the shutdown/reboot process will begin once the function is called, there won't normally be much left for your program to do. The function returns 0 if an error occurred, or 1 if successful.

uFlags

One or more of the following flags specifying how to shut down or reboot the computer:

EWX_FORCE = 4
Force any applications to quit instead of prompting the user to close them.

EWX_LOGOFF = 0
Log off the network.

EWX_POWEROFF = 8
Shut down the system and, if possible, turn the computer off.

EWX_REBOOT = 2
Perform a full reboot of the system.

EWX_SHUTDOWN = 1
Shut down the system.

dwReserved

Reserved for future versions of Windows. Always set to 0.

Example:

' Reboot the computer, forcing any open programs to close
Dim retval As Long ' return value

retval = ExitWindowsEx(EWX_REBOOT Or EWX_FORCE, 0)
If retval = 0 Then Debug.Print "Reboot attempt failed."

Category: Other
ExtFloodFill Function

Declare Function ExtFloodFill Lib "gdi32.dll" (ByVal hdc As Long, ByVal nXStart As Long, ByVal nYStart As Long, ByVal crColor As Long, ByVal fuFillType As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ExtFloodFill performs a flood fill operation on a device using that device's currently selected brush. The flood fill begins at a single point and extends in all directions until a condition is met. The flood fill can extend either until a certain boundary color is reached or while a certain color is being filled over.

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- hdc
  A handle to a device context to the device to perform the flood fill on.
- nXStart
  The x-coordinate of the point to begin the flood fill at.
- nYStart
  The y-coordinate of the point to begin the flood fill at.
- crColor
  The RGB value of the color determining the extent of the flood fill operation. Its exact interpretation depends on the
flag passed as fuFillType.

fuFillType

One of the following flags specifying how to determine the boundary of the flood fill operation:

**FLOODFILLBORDER**

Fill from the beginning point in all directions until a boundary of color crColor is reached. The flood fill will cover over any colors within the region which do not have the color of crColor.

**FLOODFILLSURFACE**

Fill from the beginning point in all directions as long as the fill-in color crColor is encountered. The boundary of the flood fill is made up of any color which is not identical to crColor.

### Constant Definitions

Const FLOODFILLBORDER = 0
Const FLOODFILLSURFACE = 1

### Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/e/extfloodfill.html).  

' Draw a solid green ellipse on window Form1 and fill  
' the area outside of it with a diagonally cross-hatched red brush.  
Dim hPen As Long, hBrush As Long  ' handles to the pen and brush to be used  
Dim hOldPen As Long, hOldBrush As Long  ' handles to the previously selected objects  
Dim retval As Long  ' return value  

' Create a solid green pen with a width of one pixel.  
hPen = [CreatePen](PS_SOLID, 0, RGB(0, 255, 0))  
' Select it for use in Form1, noting the previous pen.  
hOldPen = [SelectObject](Form1.hDC, hPen)  
' Create a blue diagonally cross-hatched brush.  
hBrush = [CreateHatchBrush](HS_DIAGCROSS, RGB(0, 0, 255))  
' Select it for use in Form1, noting the previous brush.  
hOldBrush = [SelectObject](Form1.hDC, hBrush)

' Draw an ellipse with bounding rectangle (100,150)-(350, 250)  
retval = [Ellipse](Form1.hDC, 100, 150, 350, 250)  
' Flood fill the area outside the ellipse (use a green boundary)  
' starting at the point (25,30) outside the ellipse.  
retval = [ExtFloodFill](Form1.hDC, 25, 25, RGB(0, 255, 0), FLOODFILLBORDER)

' Select the previously selected pen and brush.  
retval = [SelectObject](Form1.hDC, hOldPen)  
retval = [SelectObject](Form1.hDC, hOldBrush)  
' Delete the pen and brush to free up resources.  
retval = [DeleteObject](hPen)  
retval = [DeleteObject](hBrush)
ExtractIcon Function

Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst as Long, ByVal lpszExeFileName As String, ByVal nIconIndex As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ExtractIcon extracts a single icon from a file. This file can be an executable (.exe) file, a dynamic link library (.dll), or an icon file (.ico). Alternately, this function can also determine how many icons are stored in such a file. The icon generated by this function must be destroyed using DestroyIcon after the program has finished using it.

Return Value

If the function failed because the specified file was not found, the function returns 1. If the function failed because the icon requested by the function did not exist, the function returns 0. If the function succeeded and the number of icons in the file was requested, the function returns the number of icons stored in the file. If the function succeeded and an icon was specified, the function returns a handle to the extracted icon.

Visual Basic-Specific Issues

None.

Parameters

hInst

A handle to the instance of the application calling the function.

lpszExeFileName
The name of an .exe, .dll, or .ico to extract an icon from.

\textit{nIconIndex}

If this is -1, the function returns the number of icons stored in the specified file. If this is a non-negative number, the function extracts the icon using this value as the zero-based index (an index of 0 identifies the first icon, etc.). \textbf{Windows 95, 98, NT 4.0 or later, 2000:} If this is negative and not -1, the function extracts the icon whose resource identifier equals the absolute value of this parameter. (To extract an icon with a resource identifier of 1, \texttt{ExtractIconEx} must be used instead.)

\section*{Example}

'D This code is licensed according to the terms and conditions listed \texttt{here}.

'D Display the first icon (index 0) stored in the executable file
'C:\MyApp\Prog.exe on window Form1. The icon must be destroyed after the
'program finishes using it.
Dim hIcon As Long ' handle to the function gotten from the executable file
Dim retval As Long ' return value

' Extract the first icon stored in the aforementioned executable file.
hIcon = \texttt{ExtractIcon}(App.hInstance, "C:\MyApp\Prog.exe", 0)
' Only attempt to display the icon if we successfully extracted it.
If hIcon = 0 Then
    Debug.Print "Failed to extract the icon -- aborting."
Else
    ' Display the icon at coordinates (100, 75) on window Form1.
    retval = \texttt{DrawIcon}(Form1.hDC, 100, 75, hIcon)
    ' Although the icon's image is still visible, the icon itself is not in use.
    ' Therefore we destroy it to free up resources.
    retval = \texttt{DestroyIcon}(hIcon)
End If

\section*{See Also}

\texttt{ExtractIconEx}

\section*{Category}

\texttt{Icons}

Go back to the alphabetical Function listing.
Go back to the Reference section index.
Windows API Guide: ExtractIcon Function

Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/extracticon.html
ExtractIconEx Function

Declare Function ExtractIconEx Lib "shell32.dll" Alias "ExtractIconExA" (ByVal lpszFile As String, ByVal nIconIndex As Long, phiconLarge As Long, phiconSmall As Long, ByVal nIcons As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

ExtractIconEx extracts multiple icons from a file. This file can be an executable file (.exe), a dynamic link library (.dll), or an icon file (.ico). This function can extract both large and small icons, whose handles are placed into two arrays. Optionally, this function can also determine how many large/small icon pairs are stores in such a file. Each icon which this function extracts must be destroyed using DestroyIcon after the program has finished using it.

Return Value

If nIconIndex is set to -1, phiconLarge is set to 0, and phiconSmall is set to 0, the function returns the number of icons stored in the file specified. Otherwise, the function returns the number of icons successfully extracted from the file.

Visual Basic-Specific Issues

When passing 0 explicitly as phiconLarge or phiconSmall, the 0 must be preceeded by the ByVal keyword. See the example for a demonstration.

Parameters

lpszFile

The name of the .exe, .dll, or .ico file to extract the icons from.

nIconIndex

The zero-based index of the first icon to extract from the file. If this is -1 and both phiconLarge and phiconSmall are 0, the function returns the number of icons stored in the file. Windows 95, 98, NT 4.0 or later, 2000: If this is a negative integer and at least either phiconLarge or phiconSmall (or both) are not zero, the function first extracts the
icon whose resource identifier equals the absolute value of this parameter.

**phiconLarge**
An array which receives the handles of the large icons extracted from the file. To not extract any large icons, pass 0 as this parameter.

**phiconSmall**
An array which receives the handles of the small icons extracted from the file. To not extract any small icons, pass 0 as this parameter.

**nIcons**
The number of icons to extract from the file. Icons are extracted sequentially, beginning with the icon identified by **nIconIndex**.

**Example**

```vba
' This code is licensed according to the terms and conditions listed here.

' Extract all of the regular-sized icons from the file
' C:\MyApp\Prog.exe. Display them in a row, stretching or shrinking them to
' a width of 32 and a height of 64. Note how dynamically allocated arrays
' are used to receive the icon handles. Draw all icons on a light-gray
' background on the window Form1.
Dim hIcons() As Long ' dynamic array to receive handles to the icons
Dim numicons As Long ' number of regular icons in the file
Dim hBrush As Long ' handle to the background brush to use
Dim c As Long ' counter variable
Dim retval As Long ' return value

' Determine how many regular icons exist in the file and resize
' the array accordingly.
numicons = ExtractIconEx("C:\MyApp\Prog.exe", -1, ByVal 0, ByVal 0, 0)
If numicons = 0 Then
    Debug.Print "No icons found in the file -- aborting."
    End ' abort the program if failure occurs
End If
ReDim hIcons(0 To numicons - 1) As Long ' resize the array to hold all the handles

' Get a handle to the stock solid light gray brush to use for the background.
hBrush = GetStockObject(LTGRAY_BRUSH) ' handle to the brush

' Extract all of the icons to display.
retval = ExtractIconEx("C:\MyApp\Prog.exe", numicons, hIcons(0), ByVal 0, 0)

' Loop through each icon, displaying it as previously mentioned.
For c = 0 To numicons - 1
    ' The x coordinate equals 32 * c. The y coordinate is always 0.
    ' Display this particular icon.
    retval = DrawIconEx(Form1.hDC, 32 * c, 0, hIcons(c), 32, 64, 0, hBrush, DI_NORMAL)
    ' Now destroy this icon since we no longer are using it.
    retval = DestroyIcon(hIcons(c))
Next c
```

**See Also**

FileStreamToLocalFileTime Function

Declare Function FileStreamToLocalFileTime Lib "kernel32.dll" (lpFileTime As FILETIME, lpLocalFileTime As FILETIME) As Long

Platforms: Win32s, Win 95/98, Win NT

FileStreamToLocalFileTime converts a time from UTC time (also known as Greenwich Mean Time) to "local time" (inside the computer's selected time zone). The source and target times are stored in FILETIME structures. The function returns 1 if successful, or 0 if an error occurs.

lpFileTime
The source time and date, which are in UTC time.

lpLocalFileTime
Receives the time and date stored in lpFileTime converted into the computer's current time zone time.

Example:

' Search for all files that match "C:\MyProgram\user*.\". Display
' the creation time of each file. Since the file search functions give the file
times in UTC
' time, they must be converted to local time before they are displayed.
Dim hsearch As Long  ' handle to the file search
Dim findinfo As WIN32_FIND_DATA  ' receives info about matching files
Dim success As Long  ' will be 1 if successive searches are successful, 0 if not
Dim localtime As FILETIME  ' receives local creation time
Dim systime As SYSTEMTIME  ' receives creation time
Dim retval As Long  ' generic return value

' Begin a file search:
hsearch = FindFirstFile("C:\MyProgram\user*.*", findinfo)
If hsearch = -1 Then  ' no files match the search string
    Debug.Print "(no files matched search parameter)"
End If

' Display creation date of each file that matches the search. Note that the name
' is displayed, the next file (if any) is found, and then the loop restarts.
' This way the first file (found above) will also be displayed.
Do  ' begin loop
    ' Convert UTC FILETIME to local SYSTEMTIME and display the date:
    retval = FileStreamToLocalFileTime(findinfo.ftCreationTime, localtime)
    Debug.Print localtime
    ' Next file
End Do
retval = FileTimeToSystemTime(localtime, systime)
Debug.Print "Date:"; systime.wMonth; "-"; systime.wDay; "-"; systime.wYear

' Get the next matching file and loop if it exists:
success = FindNextFile(hsearch, findinfo)
Loop Until success = 0  ' keep looping until no more matching files are found

' Close the file search handle
retval = FindClose(hsearch)

See Also: LocalFileTimeToFileTime
Category: Time
FileTimeToSystemTime Function

Declare Function FileTimeToSystemTime Lib "kernel32.dll" (lpFileTime As FILETIME, lpSystemTime As SYSTEMTIME) As Long

Platforms: Win 32s, Win 95/98, Win NT

FileTimeToSystemTime converts a time and date stored in a FILETIME structure to an identical time and date stored in a SYSTEMTIME structure. The latter structure provides a easier way to access a date and time, whereas the former is used by Windows to identify times and dates associated with files. The data put into the SYSTEMTIME structure identifies the same time and date as the source structure does. The function returns 0 if an error occurred, or 1 if successful.

lpFileTime
The date and time, in FILETIME form, to convert.

lpSystemTime
Receives the date and time converted into SYSTEMTIME format.

Example:

' Display the date when file C:\MyProgram\datafile.txt was created. Note how CreateFile's alternate declare must be used under Win 95/98 -- see that function's page for more information. Also note how the times returned by GetFileTime need to be converted so the program can figure out what the date actually is.
Dim hfile As Long ' receives the handle to the file
Dim ctime As FILETIME ' receives creation date and time of the file
Dim atime As FILETIME ' receives last access date and time of the file
Dim wtime As FILETIME ' receives last write-to date and time of the file
Dim createtime As SYSTEMTIME ' receives a converted form of ctime
Dim retval As Long ' return value

' Get a handle to the file (note how the alternate declare is used):
hfile = CreateFileNS("C:\MyProgram\datafile.txt", GENERIC_READ, FILE_SHARE_READ, 0, OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then ' if the file could not be opened
    Debug.Print "Could not open the file C:\MyProgram\datafile.txt."
    End ' abort the program
End If

' Get the various times and dates associated with the file:
retval = GetFileTime(hfile, ctime, atime, wtime)
' Convert the creation time from a FILETIME structure to a SYSTEMTIME structure (for usability):
retval = **FileTimeToDateTime**(ctime, createtime)
' Display the creation date of the file:
Debug.Print "Creation Date:"; createtime.wMonth; ","; createtime.wDay; ","; createtime.wYear
'
' Close the file
retval = **CloseHandle**(hfile)

**See Also:** [SystemTimeToFileTime](http://www.vbapi.com/ref/f/systemtimetoftime.html)

**Category:** Time

[Go back to the Reference section index.](http://www.vbapi.com/ref/f/function_list.html)
**FillMemory Function**

Declare Sub FillMemory Lib "kernel32.dll" Alias "RtlFillMemory" (Destination As Any, ByVal Length As Long, ByVal Fill As Byte)

**Platforms**

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

**Description & Usage**

*FillMemory* fills a location in memory with a certain value. The function does this by setting each byte starting at the given memory location to the desired value. The memory location is identified by a pointer to the memory address.

**Return Value**

*FillMemory* does not return a value.

**Visual Basic-Specific Issues**

A pointer to any variable can be automatically generated merely by passing that variable as *Destination*. However, if either a String or a Long holding the desired memory address is passed, the ByVal keyword must precede it. See the example below for a demonstration.

**Parameters**

- **Destination**
  A pointer to the location in memory (often the memory address of a variable) to begin filling with a certain value.
- **Length**
  The number of memory bytes, beginning with the address identified by *Destination*, to fill.
- **Fill**
  The byte value to set each byte in the desired memory location to.
Example

' This code is licensed according to the terms and conditions listed here.

' Initialize all the elements in an array of bytes to the value 76. Also
' set each character in a 20-character string to the character "X".
Dim bytarray(0 To 9) As Byte  ' array of 10 bytes
Dim bytestring As String  ' string to fill
Dim c As Integer  ' counter variable

' Fill the memory at bytarray() in order to initialize its members to 76. Note
' that, to
' identify the pointer to bytarray()'s memory, it is passed as normal.
FillMemory bytarray(0), 10, 76  ' fill 10 bytes to the byte value 76
' Display the results to verify that it worked.
For c = 0 To 9  ' loop through each element
  Debug.Print bytarray(c);  ' each value displayed will be 76
Next c

' Now fill a 20-character string with "X" (using its ASCII code). Note how, in
Visual
' Basic, the ByVal keyword must preceed the string in this case.
bytestring = Space(20)  ' make the string 20 characters long
FillMemory ByVal bytestring, 20, Asc("X")  ' set the contents to a bunch of "X"'s
' Display the results to verify that it worked.
Debug.Print bytestring  ' will be twenty X's

See Also

ZeroMemory

Category

Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: July 25, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/f/fillmemory.html
FillRect Function

Declare Function FillRect Lib "user32.dll" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FillRect fills a rectangular area on a device using the specified brush. The outline of the rectangular area is not drawn, and the bottom and right edges of the given rectangle are not filled in (they are not considered to be part of the interior of the rectangle). Note that this function uses the brush passed to the function, so it is not necessary to use SelectObject to have the device select the brush first. The function returns 1 if successful, or 0 if an error occurred.

 hdc
 A device context to the device to fill a rectangular area of.

 lpRect
 The coordinates of the rectangular area to fill.

 hBrush
 A handle to the brush to use to fill in the rectangular area.

Example:

' Use a blue diagonal-cross hatched brush to fill in a rectangular
' area on window Form1. The rectangular area has coordinates (20,25)-(200,175).
Dim hbrush As Long  ' receives handle to the blue hatched brush to use
Dim r As RECT  ' rectangular area to fill
Dim retval As Long  ' return value

' Set the coordinates of the rectangle r
retval = SetRect(r, 20, 25, 200, 175)  ' now r = (20,25)-(200,175)

' Create a blue diagonal-cross hatched brush
hbrush = CreateHatchBrush(HS_DIAGCROSS, RGB(0, 0, 255))

' Fill in the desired rectangular area
retval = FillRect(Form1.hDC, r, hbrush)  ' fill the rectangle using the brush

' Delete the brush we created in order to free up resources
retval = DeleteObject(hbrush)

See Also: FillRgn, FrameRect
Category: Filled Shapes
FillRgn Function

Declare Function FillRgn Lib "gdi32.dll" (ByVal hdc As Long, ByVal hRgn As Long, ByVal hBrush As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FillRgn fills the area defined by a region on a device. Instead of using the device's currently selected brush, the region is filled using a brush passed to the function. The boundary of the region is not drawn; only its area is filled. The function returns 0 if an error occurred, or a non-zero value if successful.

(hdc)  A device context to the device to fill a region of.
(hRgn) A handle to the region on the device to fill.
(hBrush) A handle to the brush to use to fill the region.

Example:

' This code is licensed according to the terms and conditions listed here.

' Use the light-gray solid stock brush to fill an elliptical region on window Form1. The bounding rectangle of the ellipse is (30,20)-(150,110).
Dim hrgn As Long ' handle to the region to fill
Dim hbrush As Long ' handle to the brush to fill the region with
Dim retval As Long ' return value

' First, get a handle to the stock light-gray solid brush.
hbrush = GetStockObject(LTGRAY_BRUSH)
' Next, create the elliptical region and get a handle to it.
hrgn = CreateEllipticRgn(30, 20, 150, 110)
' Fill the region using the light-gray brush.
retval = FillRgn(Form1.hDC, hrgn, hbrush)
' Delete the region to free resources. The stock brush does not need to be deleted.
retval = DeleteObject(hrgn)

See Also: FillRect, FrameRgn
Category: Regions
FindClose Function

Declare Function FindClose Lib "kernel32.dll" (ByVal hFindFile As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FindClose terminates a file-search operation initiated by FindFirstFile. This function closes the file search handle.

hFindFile
The search handle of the file-search operation to end.

Example:

' Search for all files that match "C:\MyProgram\user*.*". Display
' the filename of each file that matches the string.
Dim hsearch As Long  ' handle to the file search
Dim findinfo As WIN32_FIND_DATA  ' receives info about matching files
Dim success As Long  ' will be 1 if successive searches are successful, 0 if not
Dim buffer As Long  ' string buffer to use to process the filename(s)
Dim retval As Long  ' generic return value

' Begin a file search:
hsearch = FindFirstFile("C:\MyProgram\user*.*", findinfo)
If hsearch = -1 Then  ' no files match the search string
    Debug.Print "(no files matched search parameter)"
    End  ' abort program
End If

' Display name of each file that matches the search. Note that the name is
' displayed, the
' next file (if any) is found, and then the loop restarts. This way the first file
' (found above) will also be displayed.
Do  ' begin loop
    ' Extract the filename from the fixed-length string:
    buffer = Left(findinfo.cFileName, InStr(findinfo.cFileName, vbNullChar) - 1)
    Debug.Print buffer  ' display this filename

    ' Get the next matching file and loop if it exists:
    success = FindNextFile(hsearch, findinfo)
Loop Until success = 0  ' keep looping until no more matching files are found

' Close the file search handle
retval = FindClose(hsearch)
See Also: FindFirstFile, FindNextFile
Category: Files
FindFirstFile Function

Declare Function FindFirstFile Lib "kernel32.dll" Alias "FindFirstFileA" (ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA) As Long

Platforms: Win 32s, Win 95/98, Win NT

FindFirstFile begins a file search and provides information about the first matching file. The function searches for files based only on a filename with wildcards (* or ?). The search only looks in a single directory for the file(s), but it will identify any directory names in that directory that match the search string. Identifying information about the file is put into the variable passed as lpFindFileData. The function returns a "search handle" which can be used to look for additional matching files (via FindNextFile), or -1 if no files match the search (or if an error occurred).

lpFileName
The file search string to look for, including the complete path. It can contain the wildcards * or ?.

lpFindFileData
Receives identifying information about the first file that matches the search string.

Example:

' Search for all files that match "C:\MyProgram\user*.*". Display
' the filename of each file that matches the string.
Dim hsearch As Long  ' handle to the file search
Dim findinfo As WIN32_FIND_DATA  ' receives info about matching files
Dim success As Long  ' will be 1 if successive searches are successful, 0 if not
Dim buffer As Long  ' string buffer to use to process the filename(s)
Dim retval As Long  ' generic return value

' Begin a file search:
 hsearch = FindFirstFile("C:\MyProgram\user*.*", findinfo)
 If hsearch = -1 Then  ' no files match the search string
   Debug.Print "(no files matched search parameter)"
   Exit ' abort program
 End If

' Display name of each file that matches the search. Note that the name is
' displayed, the
' next file (if any) is found, and then the loop restarts. This way the first file
' (found above) will also be displayed.
Do ' begin loop
  buffer = Left(findinfo.cFileName, InStr(findinfo.cFileName, vbNullChar) - 1)
  Debug.Print buffer ' display this filename
' Get the next matching file and loop if it exists:
success = FindNextFile(hsearch, findinfo)
Loop Until success = 0  ' keep looping until no more matching files are found
'
' Close the file search handle
retval = FindClose(hsearch)

See Also: FindClose, FindNextFile
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
FindNextFile Function

Declare Function FindNextFile Lib "kernel32.dll" Alias "FindNextFileA" (ByVal hFindFile As Long, lpFindFileData As WIN32_FIND_DATA) As Long

Platforms: Win 32s, Win 95/98, Win NT

FindNextFile continues a file search began by FindFirstFile. It finds and provides identifying information about the next file that matches the search string. This information is put into the variable passed as lpFindFileData. The function returns 1 if another matching file was found, or 0 if no more matching files exist (or if an error occured).

hFindFile
The handle to the file search initiated by FindFirstFile.

lpFindFileData
Receives identifying information about the next matching file that was found.

Example:

' Search for all files that match "C:\MyProgram\user*.\". Display
' the filename of each file that matches the string.
Dim hsearch As Long  ' handle to the file search
Dim findinfo As WIN32_FIND_DATA  ' receives info about matching files
Dim success As Long  ' will be 1 if successive searches are successful, 0 if not
Dim buffer As Long  ' string buffer to use to process the filename(s)
Dim retval As Long  ' generic return value

' Begin a file search:
hsearch = FindFirstFile("C:\MyProgram\user*.\", findinfo)
If hsearch = -1 Then  ' no files match the search string
    Debug.Print "(no files matched search parameter)"
    End  ' abort program
End If

' Display name of each file that matches the search. Note that the name is
' displayed, the
' next file (if any) is found, and then the loop restarts. This way the first file
' (found above) will also be displayed.
Do ' begin loop
    ' Extract the filename from the fixed-length string:
    buffer = Left(findinfo.cFileName, InStr(findinfo.cFileName, vbNullChar) - 1)
    Debug.Print buffer  ' display this filename

' Get the next matching file and loop if it exists:
success = \texttt{FindNextFile}(hsearch, findinfo)

Loop Until success = 0  ' keep looping until no more matching files are found

' Close the file search handle
retval = \texttt{FindClose}(hsearch)

\textbf{See Also:} \texttt{FindClose}, \texttt{FindFirstFile}
\textbf{Category:} Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
FindWindow Function

Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" (ByVal lpClassName As String, ByVal lpWindowName As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

FindWindow searches all windows for one which matches the window class name and/or window name. The function's searching mechanism is not case-sensitive. If you do not wish to specify one of the parameters, pass a null string for it.

Return Value

If successful, the function returns a handle to the window that was found. If no matching window could be found, or if an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

To pass a null string as one of the function's parameters, use the vbNullString constant.

Parameters

lpClassName

The name of the window class of the window to find. To ignore the window's class, specify a null string.

lpWindowName

The name of the title bar text of the window to find. To ignore the window's text, specify a null string.

Example

Search for a window called Minesweeper and flash its title bar once. We don't need to know the name of the window's class to find it, since most likely there won't be any unless Windows's Minesweeper game is running. This is all done when the user clicks button cmdFind, so to use this example, you naturally must place a command button named cmdFind on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" (ByVal lpClassName As String, _
     ByVal lpWindowName As String) As Long
Public Declare Function FlashWindow Lib "user32.dll" (ByVal hWnd As Long, ByVal bInvert As Long) As Long
Public Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)

' *** Place the following code inside a form. ***
Private Sub cmdFind_Click()
    Dim hWnd As Long    ' receives handle to the found window
    Dim retval As Long  ' generic return value

    ' Attempt to locate a window titled Minesweeper.
    hWnd = FindWindow(vbNullString, "Minesweeper")
    If hWnd = 0 Then
        Debug.Print "Minesweeper is not currently running."
    Else
        ' Flash the window's title bar on and off once.
        retval = FlashWindow(hWnd, 1)
        Sleep 500    ' pause for half a second
        retval = FlashWindow(hWnd, 0)
    End If
End Sub

See Also

FindWindowEx

Category

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Windows API Guide: FindWindow Function

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/f/findwindow.html
FindWindowEx Function

Declare Function FindWindowEx Lib "user32.dll" Alias "FindWindowExA" (ByVal hwndParent As Long, ByVal hwndChildAfter As Long, ByVal lpszClass As String, ByVal lpszWindow As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

FindWindowEx searches for a window matching a specified window class name and/or window name. The function searches all of the child windows of a given window (if desired), beginning with the windows below a specified child window. To ignore the window's class and/or title, pass a null string for those parameters.

Return Value

If successful, the function returns a handle to the found window. If no matching window could be found, or if an error occurred, the function returns zero (use GetLastError to get the error code.

Visual Basic-Specific Issues

To pass a null string as one of the function's parameters, use the vbNullString constant.

Parameters

hwndParent A handle to the parent window to search the child windows of. To search all windows, specify 0 for this parameter.

hwndChildAfter A handle to the child window specifying a place to begin searching. Searching begins with the child window immediately after this window in the Z-order. If this is 0, searching begins with the child window at the top of the Z-order.

lpszClass The name of the window class of the window to find. Specify a null string to ignore the class.

lpszWindow The name of the title bar text of the window to find. Specify a null string to ignore the window's title.
Example

Display the name of the first command button on window Form1. The buttons on a form are child windows of the window they appear in. In VB5 and VB6, buttons created in the Form Editor have class "ThunderCommandButton". This search runs when the user clicks button cmdFind, so to use this example, you must first place a command button named cmdFind on a form window.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Declare Function FindWindowEx Lib "user32.dll" Alias "FindWindowExA" (ByVal hwndParent As Long, ByVal hwndChildAfter As Long, ByVal lpszClass As String, ByVal lpszWindow As String) As Long
Public Declare Function GetWindowTextLength Lib "user32.dll" Alias "GetWindowTextLengthA" (ByVal hWnd As Long) As Long
Public Declare Function GetWindowText Lib "user32.dll" Alias "GetWindowTextA" (ByVal hWnd As Long, ByVal lpString As String, ByVal nMaxCount As Long) As Long

' *** Place the following code inside the form window. ***

Private Sub cmdFind_Click()
    Dim hWnd As Long       ' handle to the found window (the command button)
    Dim slength As Long    ' length of the found window's text
    Dim wintext As String  ' holds the window's text
    Dim retval As Long     ' return value

    ' Find the "topmost" command button on Form1. Begin searching at the top.
    hWnd = FindWindowEx(Form1.hWnd, 0, "ThunderCommandButton", vbNullString)
    If hWnd = 0 Then
        Debug.Print "No command buttons of class ThunderCommandButton were found."
    Else
        ' Get the text displayed in the button.
        slength = GetWindowTextLength(hWnd)
        wintext = Space(slength + 1)
        retval = GetWindowText(hWnd, wintext, slength + 1)
        ' Remove the terminating null and display the result.
        wintext = Left(wintext, slength)
        Debug.Print "The command button's name is: "; wintext
    End If
End Sub

See Also

FindWindow

FlashWindow Function

Declare Function FlashWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal bInvert As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FlashWindow flashes a window one step. Flashing is where the title bar of the window is switched from an active to inactive look (or vice versa) to get the user's attention. Normally this is done multiple times, instead of just once. When you are done flashing, be sure to call the function again, this time with bInvert set to 0. The function returns 0 if the window's look was inactive before flashing, or 1 if its look was active.

hwnd

The handle of the window to flash one step.

bInvert

Specifies how to flash. If non-zero, switches the title bar from an active to inactive look (or vice versa). If zero, restores the window to its normal look.

Example:

' Flash Form1 five times to get the user's attention
Dim c As Integer, retval As Long  ' counter variable & return value
For c = 1 To 10  ' flash on five times, off five times
    retval = FlashWindow(Form1.hWnd, 1)  ' toggle the look of the window
    Sleep 500  ' halt execution for 500 milliseconds (1/2 minute)
Next c
retval = FlashWindow(Form1.hWnd, 0)  ' make sure the window looks normal

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
FrameRect Function

Declare Function FrameRect Lib "user32.dll" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FrameRect draws a one-pixel-wide frame around a rectangle on a device using a given brush. This frame is equivalent to what the edge of a filled rectangle (using FillRect) would be. Note that this function uses the brush passed to the function, so it is not necessary to use SelectObject to have the device select the brush first. The function returns 1 if successful, or 0 if an error occurred.

hdc
   A device context to the device to draw the rectangular frame on.
lpRect
   The rectangle that defines the rectangular frame to draw.
hBrush
   A handle to the brush to use to draw the rectangular frame.

Example:

' Use a blue diagonal-cross hatched brush to draw a rectangular frame on window Form1. The rectangular frame has coordinates (20,25)-(200,175).
Dim hbrush As Long  ' receives handle to the blue hatched brush to use
Dim r As RECT  ' rectangular area to frame
Dim retval As Long  ' return value

' Set the coordinates of the rectangle r
retval = SetRect(r, 20, 25, 200, 175)  ' now r = (20,25)-(200,175)
' Create a blue diagonal-cross hatched brush
hbrush = CreateHatchBrush(HS_DIAGCROSS, RGB(0, 0, 255))
' Fill in the desired rectangular area
retval = FrameRect(Form1.hDC, r, hbrush)  ' frame the rectangle using the brush
' Delete the brush we created in order to free up resources
retval = DeleteObject(hbrush)

See Also: FillRect, FrameRgn
Category: Filled Shapes

Go back to the alphabetical Function listing.
Windows API Guide: FrameRect Function

Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at http://www.vbapi.com/ref/f/framerect.html
FrameRgn Function

Declare Function FrameRgn Lib "gdi32.dll" (ByVal hdc As Long, ByVal hRgn As Long, ByVal hBrush As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

FrameRgn draws a frame (border) around a given region on a device using the specified brush. The device's currently selected brush is not used. The width and height of the drawn frame are also specified by the function. The function returns 0 if an error occurred, or a non-zero value if successful.

hdc
A device context to the device to draw on.

hRgn
A handle to the region to draw the frame of.

hBrush
A handle to the brush to use to draw the frame.

nWidth
The width in pixels of vertical brush strokes to use to draw the frame.

nHeight
The height in pixels of horizontal brush strokes to use to draw the frame.

Example:

' Draw a frame around an elliptical region on window Form1. The frame will have a width of 5 and a height of 3. The region has bounding rectangle (20,30)-(220,180).
' A green diagonally cross-hatched brush is used.
Dim hRgn As Long ' handle to the region to frame
Dim hBrush As Long ' handle to the green diagonally cross-hatched brush
Dim retval As Long ' generic return value

' Create the elliptical region and the brush.
hRgn = CreateEllipticRgn(20, 30, 220, 180) ' elliptical region
hBrush = CreateHatchBrush(HS_DIAGCROSS, RGB(0, 255, 0)) ' brush

' Frame the region using the created brush.
retval = FrameRgn(Form1.hDC, hRgn, hBrush, 5, 3) ' frame width = 5, height = 3

' Delete the region and brush to free up resources.
retval = DeleteObject(hRgn)
retval = DeleteObject(hBrush)
GetActiveWindow Function

Declare Function GetActiveWindow Lib "user32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

GetActiveWindow returns a handle to your program's currently active window. This only works with windows created by your application -- in other words, it won't find the active window of other programs. If your program is in the background, the function will get the window that would be active if the program were active. If an error occurs, or if there is no active window to your program, the function instead returns 0.

Example:

' Use FlashWindow to flash the title bar of the program's currently active window once.
Dim hactive As Long  ' handle to the active window
Dim retval As Long   ' return value

hactive = GetActiveWindow() ' get the handle of the program's active window
' The next three lines flash the window's title bar
retval = FlashWindow(hactive, 1): Sleep 250
retval = FlashWindow(hactive, 1): Sleep 250
retval = FlashWindow(hactive, 0)

See Also: GetForegroundWindow, GetWindow, SetActiveWindow

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getactivewindow.html
GetArcDirection Function

Declare Function GetArcDirection Lib "gdi32.dll" (ByVal hdc As Long)

Platforms: Win 32s, Win 95/98, Win NT

GetArcDirection determines the direction in which arcs are drawn on a graphics-capable device. Arcs can be drawn either clockwise or counterclockwise from the starting point to the ending point. Although this function is supported in Win 95/98, that platform ignores the setting and always draws arcs counterclockwise! The function returns 0 if an error occurred, or exactly one of the following flags specifying which direction arcs on the device will be drawn:

AD_CLOCKWISE = 2
   Arcs are drawn clockwise from the starting point to the ending point.
AD_COUNTERCLOCKWISE = 1
   Arcs are drawn counterclockwise from the starting point to the ending point.

hdc
   The device context of the device to find the arc-drawing direction of.

Example:

' Display which direction window Form1 draws arcs.
Dim arcdir As Long ' receives arc direction
arcdir = GetArcDirection(Form1.hDC) ' get the arc direction for Form1
If arcdir = AD_CLOCKWISE Then
   Debug.Print "Form1 draws arcs clockwise."
ElseIf arcdir = AD_COUNTERCLOCKWISE Then
   Debug.Print "Form1 draws arcs counterclockwise."
End If

See Also: AngleArc, Arc, ArcTo, SetArcDirection
Category: Lines & Curves
GetAsyncKeyState Function

Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As Integer

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetAsyncKeyState determines whether a certain key is currently pressed and whether that key has been pressed since the last call to the function. This function fails if the thread calling it does not currently have the input focus.

Return Value

If the function fails (if the current thread does not have the input focus), the function returns 0. If the &H8000 bit of the return value is set, the key has been pressed at least once since the last time the thread called GetAsyncKeyState. If the &H1 bit of the return value is set, the key is currently pressed down.

Visual Basic-Specific Issues

None.

Parameters

**vKey**

The virtual-key code of the key to check. **Windows NT, 2000**: This could also be one of the following flags which distinguish between the left and right Ctrl, Alt, and Shift keys:

- **VK_LSHIFT**
  - The left Shift key.
- **VK_RSHIFT**
  - The right Shift key.
- **VK_LCONTROL**
  - The left Ctrl key.
VK_RCONTROL
    The right Ctrl key.
VK_LMENU
    The left Alt key.
VK_RMENU
    The right Alt key.

Constant Definitions

Const VK_LSHIFT = &HA0
Const VK_RSHIFT = &HA1
Const VK_LCONTROL = &HA2
Const VK_RCONTROL = &HA3
Const VK_LMENU = &HA4
Const VK_RMENU = &HA5

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetAsyncKeyState Lib "user32.dll" (ByVal vKey As Long) As Integer

' Determine whether the Q key has been pressed or not since the last call to the function (assuming this example code has already been run earlier).
' The code runs when button Command1 is pressed.
Private Sub Command1_Click()
    Dim keystate As Integer  ' state of the Q key

    ' Get the state of the Q key as returned by the function.
    ' (vbKeyQ is a VB-defined constant for Q's virtual-key code)
    keystate = GetAsyncKeyState(vbKeyQ)
    ' Check the &H8000 bit of the return value.
    If keystate And &H8000 Then
        Debug.Print "The Q key has been pressed since the last check."
    Else
        Debug.Print "The Q key has not been pressed since the last check."
    End If
End Sub

See Also

GetKeyState
GetBrushOrgEx Function

Declare Function GetBrushOrgEx Lib "gdi32.dll" (ByVal hdc As Long, lpPoint As POINT_TYPE) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetBrushOrgEx determines the origin point for using a brush on a given device is. All brushes are stored as an 8x8 pixel block, but Windows can offset the "origin" point determining the adjustment of the brush fill is. For example, an origin of (2,3) would shift the fill pattern design 2 pixels right and 3 pixels down. The current brush origin is put into the variable passed as lpPoint. The function returns 1 if successful, or 0 if an error occured.

hdc
A device context to the device to find the brush origin point of.

lpPoint
Receives the (x,y) coordinate pair of the device's brush origin point.

Example:

' Display the brush origin point for window Form1.
Dim brushorg As POINT_TYPE ' receives brush origin point
Dim retval As Long ' return value

' Determine the brush origin and display its coordinates:
retval = GetBrushOrgEx(Form1.hDC, brushorg)
Debug.Print "All brush designs are offset"; brushorg.x; "pixels right ";
Debug.Print "and"; brushorg.y; "pixels downward."

See Also: SetBrushOrgEx
Category: Brushes
GetCapture Function

Declare Function GetCapture Lib "user32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetCapture identifies which window currently has captured the mouse input, if any. However, this function only works if a window owned by the calling thread; if another program's window has captured the mouse, this function will not work.

Return Value

If a window owned by the calling thread has captured the mouse, the function returns a handle to that window. If no thread window has captured the mouse, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

None.

Constant Definitions

Example

If a window owned by the program has captured the mouse input, display its caption. If not, inform the user that no program window has captured the mouse. The capture is checked when the user clicks button Command1. To use this example, you must first place a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function GetCapture Lib "user32.dll" () As Long
Public Declare Function GetWindowText Lib "user32.dll" Alias "GetWindowTextA" (ByVal hWnd As Long, ByVal lpString As String, ByVal nMaxCount As Long) As Long
Public Declare Function GetWindowTextLength Lib "user32.dll" Alias "GetWindowTextLengthA" (ByVal hWnd As Long) As Long

*** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim hWndCapture As Long  ' handle to the window that captured the mouse
    Dim wintext As String    ' receives the text of the capturing window
    Dim slength As Long      ' the length of that string

    ' First, see if a thread window has even captured the mouse.
    hWndCapture = GetCapture()
    If hWndCapture = 0 Then
        Debug.Print "This program has not captured the mouse."
    Else
        ' Get the capturing window's text and display it.
        slength = GetWindowTextLength(hWndCapture) + 1
        wintext = Space(slength)
        wintext = Left(wintext, slength)
        Debug.Print "The window " & Chr(34) & wintext & Chr(34) & " has captured the mouse."
    End If
End Sub

See Also
ReleaseCapture, SetCapture

Category
Mouse

Back to the Function list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
GetClassInfo Function

Declare Function GetClassInfo Lib "user32.dll" Alias "GetClassInfoA" (ByVal
hInstance As Long, ByVal lpClassName As String, lpWndClass As WNDCLASS) As Long

Platforms

- **Windows 95**: Supported but Obsolete; use GetClassInfoEx instead.
- **Windows 98**: Supported but Obsolete; use GetClassInfoEx instead.
- **Windows NT**: Requires Windows NT 3.1 or later but Obsolete in Windows NT 3.5 or later; use GetClassInfoEx instead.
- **Windows 2000**: Supported but Obsolete; use GetClassInfoEx instead.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetClassInfo retrieves most of the information associated with a window class. The information is placed into the structure passed as lpWndClass.

Return Value

If an error occured, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hInstance**
  A handle to the instance which owns the window class, or 0 if the window class is defined by the operating system.
- **lpClassName**
  The name of the window class to retrieve information about.
- **lpWndClass**

Receives the information about the window class.

Example

' This code is licensed according to the terms and conditions listed here.

' Draw the icon and cursor from the window class to which window
' Form1 belongs. The two images are drawn on Form1.
Dim classinfo As WNDCLASS  ' receives the class information
Dim classname As String  ' receives the name of the window class
Dim slength As Long  ' the length of the window class's name
Dim retval As Long  ' return value

' First, get the name of the window class to which Form1 belongs.
classname = Space(255)  ' make enough room in the buffer
slength = GetClassName(Form1.hWnd, classname, 255)  ' get the name
classname = Left(classname, slength)  ' remove the empty space

' Get the information about the window class. Since this is a Visual Basic-
' generated window, its window class belongs to the application.
retval = GetClassInfo(App.hInstance, classname, classinfo)

' Now draw the window class's icon and cursor on window Form1.
' Draw the icon.
retval = DrawIconEx(Form1.hDC, 0, 0, classinfo.hIcon, 0, 0, 0, 0, DI_NORMAL)
' Draw the cursor. If it's animated, draw only the first frame of it.
retval = DrawIconEx(Form1.hDC, 50, 0, classinfo.hCursor, 0, 0, 0, 0, DI_NORMAL)

See Also

GetClassInfoEx, GetClassLong

Category

Window Classes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 21, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
GetClassInfoEx Function

Declare Function GetClassInfoEx Lib "user32.dll" Alias "GetClassInfoExA" (ByVal hinst As Long, ByVal lpszClass As String, lpwcx As WNDCLASSEX) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Not Supported.

Description & Usage

GetClassInfoEx retrieves all of the information associated with a window class. The information is placed into the structure passed as lpwcx.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- hinst
  A handle to the instance which owns the window class, or 0 if the window class is defined by the operating system.
- lpszClass
  The name of the window class to retrieve information about.
- lpwcx
  Receives the information about the window class.
Example

' This code is licensed according to the terms and conditions listed here.

' Draw the regular icon and small icon from the window class to which ' window Form1 belongs. The two icons are drawn on Form1.
Dim classinfo As WNDCLASSEX  ' receives the class information
Dim classname As String  ' receives the name of the window class
Dim slength As Long  ' the length of the window class's name
Dim retval As Long  ' return value

' First, get the name of the window class to which Form1 belongs.
classname = Space(255)  ' make enough room in the buffer
slength = GetClassName(Form1.hWnd, classname, 255)  ' get the name
classname = Left(classname, slength)  ' remove the empty space

' Get the information about the window class. Since this is a Visual Basic- ' generated window, its window class belongs to the application.
retval = GetClassInfoEx(App.hInstance, classname, classinfo)

' Now draw the window class's regular and small icons on window Form1.
' Draw the regular icon.
retval = DrawIconEx(Form1.hDC, 0, 0, classinfo.hIcon, 0, 0, 0, 0, DI_NORMAL)
' Draw the small icon.
retval = DrawIconEx(Form1.hDC, 50, 0, classinfo.hIconSm, 0, 0, 0, 0, DI_NORMAL)

See Also

GetClassInfo, GetClassLong

Category

Window Classes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 21, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getclassinfoex.html
GetClassLong Function

Declare Function GetClassLong Lib "user32.dll" Alias "GetClassLongA" (ByVal hWnd As Long, ByVal nIndex As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetClassLong retrieves a single 32-bit value from the information about the window class to which the specified window belongs. The class's properties may not necessarily match perfectly with the actual properties of the window. This function can also retrieve a 32-bit value from the extra memory area associated with the window class.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the desired 32-bit value.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to a window which belongs to the class to read a property of.

nIndex

To retrieve a 32-bit value from the class's extra memory, set this to the zero-based offset of the byte to begin reading at. Valid values are 0 to the number of bytes of extra memory minus 4, inclusive. To retrieve a 32-bit property of the class, set this to one of the following flags specifying which 32-bit value to retrieve from the window class:

GCW_ATOM
Retrieve the atom which identifies the window class.

GCL_CBCLSEXTRA
Retrieve the size in bytes of the extra memory associated with the window class.

GCL_CBWNDEXTRA
Retrieve the size in bytes of the extra memory associated with each window belonging to the window class.

GCL_HBRBACKGROUND
Retrieve a handle to the brush used to paint the backgrounds of windows belonging to the class.

GCL_HCURSOR
Retrieve a handle to the cursor associated with the class.

GCL_HICON
Retrieve a handle to the icon associated with the class.

GCL_HICONSM
Retrieve a handle to the small icon associated with the class.

GCL_HMODULE
Retrieve a handle to the module which registered the class.

GCL_MENUNAME
Retrieve a pointer to the string identifying the name of the menu resource associated with the class.

GCL_STYLE
Retrieve the window styles associated with the class.

GCL_WNDPROC
Retrieve a pointer to the WindowProc hook function which acts as the window procedure for windows belonging to the window class.

Constant Definitions

Const GCW_ATOM = -32
Const GCL_CBCLSEXTRA = -20
Const GCL_CBWNDEXTRA = -18
Const GCL_HBRBACKGROUND = -10
Const GCL_HCURSOR = -12
Const GCL_HICON = -14
Const GCL_HMODULE = -16
Const GCL_MENUNAME = -8
Const GCL_STYLE = -26
Const GCL_WNDPROC = -24

Example

When the user clicks button Command1, draw the icon associated with the parent form window's class in the corner of the client area. This might not be the icon actually displayed by the window. Naturally, to use this example, you must first place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetClassLong Lib "user32.dll" Alias "GetClassLongA" (ByVal hWnd As Long, _
Windows API Guide: GetClassLong Function

    ByVal nIndex As Long) As Long
    Public Const GCL_HICON = -14
    Public Declare Function DrawIcon Lib "user32.dll" (ByVal hDC As Long, ByVal x As Long, ByVal y As Long, ByVal hIcon As Long) As Long

' Place the following code inside a form window. ***

Private Sub Command1_Click ()
    Dim hIcon As Long ' handle to the class's icon
    Dim retval As Long ' return value

    ' Retrieve a handle to the class's icon.
    hIcon = GetClassLong(Me.hWnd, GCL_HICON)
    ' Draw that icon at coordinate (0,0) on Form1.
    retval = DrawIcon(Me.hDC, 0, 0, hIcon)
End Sub

See Also

GetWindowLong, SetClassLong

Category

Window Classes

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getclasslong.html
GetClassName Function

Declare Function GetClassName Lib "user32.dll" Alias "GetClassNameA"
(ByVal hWnd As Long, ByVal lpClassName As String, ByVal nMaxCount As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetClassName retrieves the name of the window class to which a window belongs. The name of the class is placed into the string passed as lpClassName.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of characters copied into the string passed as lpClassName.

Visual Basic-Specific Issues

None.

Parameters

- **hWnd**
  A handle to the window to get the name of the window class of.
- **lpClassName**
A string which receives the name of the window class. This must first be initialized to a sufficient length to receive the string.

\textit{nMaxCount}

The size in bytes of the string passed as \textit{lpClassName}.

\section*{Example}

' This code is licensed according to the terms and conditions listed \texttt{here}.

' Display the name of the window class to which window Form1 belongs.
Dim classname As String  ' receives the name of the class
Dim slength As Long  ' length of the string retrieved

' Make room in the string to receive the information.
classname = Space(255)  ' much more than enough room
' Get the name of the window class.
slength = \texttt{GetClassName}(Form1.hWnd, classname, 255)
' Extract the useful information from the string and display it.
classname = Left(classname, slength)  ' remove empty space
Debug.Print "Form1's window class is: "; classname

\section*{Category}

Window Classes

\url{http://www.vbapi.com/ref/g/getclassname.html}
GetClipCursor Function

Declare Function GetClipCursor Lib "user32.dll" (lprc As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetClipCursor finds the current confinement rectangle of the mouse cursor. The mouse cursor is confined by using ClipCursor. The cursor is confined inside this rectangle -- even SetCursorPos cannot free it. If there is no apparent confinement rectangle, it is actually the size of the screen. The coordinates of the rectangle is put into lprc. The function returns 0 if an error occurred, or 1 if it succeeded.

lprc Receives the coordinates of the confinement rectangle.

Example:

' Display the coordinates of the confinement rectangle.
Dim r As RECT ' receives coordinates of rectangle
Dim retval As Long ' return value

retval = GetClipCursor(r) ' read the coordinates and put them into r
Debug.Print r.Left; r.Top ' display upper-left (x,y) pair
Debug.Print r.Right; r.Bottom ' display lower-right (x,y) pair

See Also: ClipCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
GetComputerName Function

Declare Function GetComputerName Lib "kernel32.dll" Alias "GetComputerNameA" (ByVal lpBuffer As String, nSize As Long) As Long

Platforms: Win 95/98, Win NT

GetComputerName reads the name of the user's computer. The name is put into the string variable passed as lpBuffer. The function returns 0 if an error occurred or 1 if successful.

lpBuffer
   A string large enough to hold the returned computer name terminated by a null character.

nSize
   The length in characters of lpBuffer.

Example:

' Display the computer's name
Dim compname As String, retval As Long ' string to use as buffer & return value
compname = Space(255) ' set a large enough buffer for the computer name
retval = GetComputerName(compname, 255) ' get the computer's name
' Remove the trailing null character from the strong
compname = Left(compname, InStr(compname, vbNullChar) - 1)
Debug.Print compname ' display name

Category: System Information

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetCurrencyFormat Function

Declare Function GetCurrencyFormat Lib "kernel32.dll" Alias "GetCurrencyFormatA"
(ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpValue As String, lpFormat As Any, ByVal lpCurrencyStr As String, ByVal cchCurrency As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetCurrencyFormat formats a currency value for display. By default, the function formats the currency according to the specified locale's settings. However, custom formatting preferences can instead be used. The end result of GetCurrencyFormat is a currency amount displayed according to the user's preferences.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of characters copied into the string passed as lpNumberStr, not including the terminating null character.

Visual Basic-Specific Issues

When passing 0 for the lpFormat parameter, the expression ByVal CLng(0) must be used. See the example code for a demonstration of this.

Parameters

Locale

The locale identifier of the locale to format the currency value according to. This identifier could be generated by the MAKELCID macro. Alternatively, this could be one of the following flags specifying a locale:

- LOCALE_SYSTEM_DEFAULT
  The system's default locale.
- LOCALE_USER_DEFAULT
  The user's default locale.
GetCurrencyFormat Function

**dwFlags**

If no structure is passed as *lpFormat*, this parameter determines the settings used to format the currency value. If this is 0, the current locale settings are used. Or, this could be the following flag:

LOCALE_NOUSEROVERRIDE

Use the system's default settings for the locale, regardless of any modifications the user may have made to it.

**lpValue**

A string containing the number to format. The only allowable characters in this string are the digits 0-9 and at most a single decimal point character (.). If the number is negative, the first character in the string must be a minus sign character (-). Any other characters are invalid. Notice how you must not include a currency symbol!

**lpFormat**

To override the locale's formatting settings, pass a **CURRENCYFMT** structure that contains the appropriate formatting information. To use the locale's settings instead, pass 0 for this parameter.

**lpCurrencyStr**

String that receives the null-terminated formatted currency string. This string must have enough room to receive the string.

**cchCurrency**

The number of characters in the string passed as *lpCurrencyStr*.

### Constant Definitions

```
Const LOCALE_SYSTEM_DEFAULT = &H400
Const LOCALE_USER_DEFAULT = &H800
Const LOCALE_NOUSEROVERRIDE = &H80000000
```

### Example

```
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type CURRENCYFMT
    NumDigits As Long
    LeadingZero As Long
    Grouping As Long
    lpDecimalSep As String
    lpThousandSep As String
    NegativeOrder As Long
    PositiveOrder As Long
    lpCurrencySymbol As String
End Type
Public Declare Function GetCurrencyFormat Lib "kernel32.dll" Alias "GetCurrencyFormatA" _
    (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpValue As String, _
    lpFormat As Any, ByVal lpCurrencyStr As String, ByVal cchCurrency As Long) As Long
Const LOCALE_USER_DEFAULT = &H800
'
' Display the amount of $1,234,567.89 according to two formatting rules.
' 1. Use the format specified by the current user locale.
```
' 2. Use a custom format specified by a structure passed to the function.
Dim cft As CURRENCYFMT ' custom formatting settings
Dim formatted As String ' receives the formatted number strings
Dim strlen As Long ' the length of the formatted string

' Display the value formatted according to the current locale.
formatted = Space(256)
strlen = GetCurrencyFormat(LOCALE_USER_DEFAULT, 0, "1234567.89", ByVal CLng(0), _
    formatted, Len(formatted))
formatted = Left(formatted, strlen)
Debug.Print "User locale format: "; formatted

' Now display according the format we specify below.
With cft
    ' Display three digits after the decimal point (like in US gasoline prices).
    .NumDigits = 3
    ' Display zeros after the decimal point.
    .LeadingZero = 1
    ' Group every three digits to the left of the decimal.
    .Grouping = 3
    ' Use a comma to as the decimal point (like they do in France and Spain).
    .lpDecimalSep = ",,"
    ' Likewise, use a period as the grouping separator.
    .lpThousandSep = ".,"
    ' For negative values, place it in parentheses and put the $ sign after the
digits.
    .NegativeOrder = 4
    ' For positive values, place the $ sign after the digits.
    .PositiveOrder = 1
    ' Use the $ sign to represent the currency.
    .lpCurrencySymbol = "$"
End With
formatted = Space(256)
strlen = GetCurrencyFormat(LOCALE_USER_DEFAULT, 0, "-1234567.89", cft, formatted, _
    Len(formatted))
formatted = Left(formatted, strlen)
Debug.Print "Custom format: "; formatted

See Also

GetNumberFormat

Category

National Language Support

Back to the Function list.
Back to the Reference section.

GetCursor Function

Declare Function GetCursor Lib "user32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

GetCursor finds the handle to the mouse cursor currently in use. This is the cursor that is being used to represent the mouse pointer on the screen. The function returns a handle to the cursor picture if successful, or returns 0 if an error occurs.

Example:

' Display the hourglass for three seconds, then restore ' the mouse cursor to whatever it was originally.
Dim holdcursor As Long  ' receives handle to the original cursor
Dim hcursor As Long  ' receives handle to the hourglass (wait) cursor
Dim retval As Long  ' throw-away return value

holdcursor = GetCursor()  ' get the handle of the current mouse cursor
hcursor = LoadCursor(0, IDC_WAIT)  ' load the hourglass cursor
retval = SetCursor(hcursor)  ' set the cursor to the hourglass
Sleep 3000  ' wait for three seconds
retval = SetCursor(holdcursor)  ' restore the original cursor

See Also: SetCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetCursorPos Function

Declare Function GetCursorPos Lib "user32.dll" (lpPoint As POINT_TYPE) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetCursorPos reads the current position of the mouse cursor. The x and y coordinates of the cursor (relative to the screen) are put into the variable passed as lpPoint. The function returns 0 if an error occurred or 1 if it is successful.

lpPoint

Receives the x and y coordinates of the mouse cursor.

Example:

' Display the coordinates of the mouse cursor
Dim coord As POINT_TYPE  ' receives coordinates of cursor
Dim retval As Long  ' return value

retval = GetCursorPos(coord)  ' read cursor location
Debug.Print "The mouse is at:"; coord.x; coord.y

See Also: SetCursorPos
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetDateFormat Function

Declare Function GetDateFormat Lib "kernel32.dll" Alias "GetDateFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, lpDate As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpDateStr As String, ByVal cchDate As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetDateFormat formats a string to display a date according to a locale's settings. The date can be formatted using either a predefined format or a custom format specified in the parameter list. The string generated by this function can be used to present a more human-readable way to display a date.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the size in bytes of the string placed into the variable passed as lpDateStr.

Visual Basic-Specific Issues

When passing 0 for the lpFormat parameter, the expression CLng(0) must be used.

Parameters

Locale

The identifier of the locale to use to format the string as necessary. If this is 0, the locale of the calling thread is used. This can also be one of the following flags specifying a default locale:

- LOCALE_SYSTEM_DEFAULT
  - The system's default locale.
- LOCALE_USER_DEFAULT
The user's default locale.

**dwFlags**
A combination of the following flags specifying how to format the date string. If a format string is passed as *lpFormat*, this parameter must be set to 0.

- **LOCALE_NOUSEROVERRIDE**
  Use the system's default date format for the specified locale, ignoring any changes to those defaults which the user may have selected.

- **LOCALE_USE_CP_ACP**
  Use the system's ANSI code page for string translation instead of the locale's code page.

- **DATE_SHORTDATE**
  Format the date string using the short date format.

- **DATE_LONGDATE**
  Format the date string using the long date format.

- **DATEYEARMONTH**
  Format the date string using the year/month date format.

- **DATE_USE_ALT_CALENDAR**
  If an alternate calendar exists, use the default date format of that calendar to format the date string.

- **DATE_LTRREADING**
  Add marks for left-to-right reading layout.

- **DATE_RTLREADING**
  Add marks for right-to-left reading layout.

**lpDate**
The date to format as a string. The members of the structure which specify the time are ignored.

**lpFormat**
The format template string used to generate the date string. To use one of the predefined formats, this parameter must be 0. In a format template string, the following series of characters stand for the following components of the date:

- **d**
  The day of the month as digits without a leading zero for single-digit days.

- **dd**
  The day of the month as digits with a leading zero for single-digit days.

- **ddd**
  The three-letter abbreviation for the name of the day of the week.

- **dddd**
  The full name of the day of the week.

- **M**
  The month as digits without a leading zero for single-digit months.

- **MM**
  The month as digits with a leading zero for single-digit months.

- **MMM**
  The three-letter abbreviation for the name of the month.

- **MMMM**
  The full name of the month.

- **y**
  The last two digits of the year without a leading zero for years between ??00-??09.

- **yy**
  The last two digits of the year with a leading zero for years between ??00-??09.

- **yyyy**
  All four digits of the year.
The period/era string, if defined for the locale. If it is not defined, this string is ignored.

Any spaces appearing in the template string appear verbatim in the formatted string. To place any other "fixed" characters or text in the format string, you must enclose the literal text in single quotation marks.

lpDateStr
Receives the formatted date string. This must initially be sufficiently long to receive the string.

cchDate
The length of the string passed as lpDateStr.

Constant Definitions

Const LOCALE_SYSTEM_DEFAULT = &H400
Const LOCALE_USER_DEFAULT = &H800
Const LOCALE_NOUSEROVERRIDE = &H80000000
Const LOCALE_USE_CP_ACP = &H40000000
Const DATE_SHORTDATE = &H1
Const DATE_LONGDATE = &H2
Const DATE_USE_ALT_CALENDAR = &H4
Const DATE_YEARMONTH = &H8
Const DATE_LTRREADING = &H10
Const DATE_RTLREADING = &H20

Example

' This code is licensed according to the terms and conditions listed here.

' Display today's date first in the default Long Date format and ' then in the standard HTTP date format.
Dim today As SYSTEMTIME  ' today's date and time
Dim datestr As String  ' receives the formatted date string
Dim strlen As Long  ' length of the buffer for the formatted date string

' Get today's date and time in the local time zone.
GetLocalTime today

' Make sufficient room in the buffer to receive the date string.
datestr = Space(255)
' Format today's date as a Long Date.
strlen = GetDateFormat(0, DATE_LONGDATE, today, CLng(0), datestr, Len(datestr))
' Remove the empty space from the formatted date string.
datestr = Left(datestr, strlen)
' Display today's date as a Long Date.
Debug.Print "Today is "; datestr

' Now make sufficient room once again.
datestr = Space(255)
' Format today's date in the format used in HTTP.

strlen = GetDateFormat(0, 0, today, "ddd', ' dd MMM yyyy", datestr, Len(datestr))
' Remove the empty space from the formatted string.
datestr = Left(datestr, strlen)
' Display today's date in the HTTP-style format.
Debug.Print "Today is "; datestr

See Also

GetTimeFormat

Category

National Language Support

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: January 1, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getdateformat.html
GetDC Function

Declare Function GetDC Lib "user32.dll" (ByVal hWnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetDC returns the device context (DC) of a window or other device, given the object's handle. When you are finished using the device context, you should use ReleaseDC to free up system resources. If you try to get the device context of something that is not a device (i.e., pass the function a handle to a file) or another error occurs, the function will instead return 0. Do not use DeleteDC to destroy the device context when you are done.

hWnd
   The handle of the object or device to get the device context of.

Example:

' Find the device context of the desktop
Dim deskhwnd As Long  ' handle to the desktop
Dim deskhdc As Long  ' device context to the desktop
Dim retval As Long  ' return value

deskhwnd = GetDesktopWindow()  ' get the desktop's handle
deskhdc = GetDC(deskhwnd)  ' get the desktop's device context

' You could put any code that works with the desktop here
retval = ReleaseDC(deskhwnd, deskhdc)  ' free up resources associated with the device context

See Also: CreateDC, ReleaseDC
Category: Devices

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetDesktopWindow Function

Declare Function GetDesktopWindow Lib "user32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

GetDesktopWindow returns a handle to the desktop window. The desktop window is the window that makes up the desktop of the computer -- that is, the screen. If the function fails, it will return 0 instead of the handle.

Example:

' Find the device context of the desktop
' The handle to the desktop is needed to find the device context
Dim deskhwnd As Long  ' handle to the desktop
Dim deskhdc As Long  ' device context to the desktop
Dim retval As Long  ' return value

deskhwnd = GetDesktopWindow()  ' get the desktop's handle
deshdc = GetDC(deskhwnd)  ' get the desktop's device context

' You could put any code that works with the desktop here
retval = ReleaseDC(deskhwnd, deskhdc)  ' free up resources associated with the device context

Category: Windows
GetDiskFreeSpace Function

Declare Function GetDiskFreeSpace Lib "kernel32.dll" Alias "GetDiskFreeSpaceA" (ByVal lpRootPathName As String, lpSectorsPerCluster As Long, lpBytesPerSector As Long, lpNumberOfFreeClusters As Long, lpTotalNumberOfClusters As Long) As Long

Platforms

- **Windows 95**: Supported but Obsolete with OEM Service Release 2 (OEM2) or later; use GetDiskFreeSpaceEx instead.
- **Windows 98**: Supported but Obsolete; use GetDiskFreeSpaceEx instead.
- **Windows NT**: Requires Windows NT 3.1 or later but obsolete in Windows NT 4.0 or later; use GetDiskFreeSpaceEx instead.
- **Windows 2000**: Supported but Obsolete; use GetDiskFreeSpaceEx instead.
- **Windows CE**: Not Supported.

Description & Usage

GetDiskFreeSpace retrieves information about the amount of space on a disk. This information includes the number of sectors in each cluster, the number of bytes in each sector, the number of free clusters, and the total number of clusters. Due to the limitations of the 32-bit integer data type, this function only works properly with disks with a capacity less than 2 MB. The replacement function GetDiskFreeSpaceEx does not have this limitation.

Return Value

If an error occurred, the function returns 0 (use GetLastError to determine the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None

Parameters
**lpRootPathName**  
The root directory of the disk to get information on, such as "c:\" or "a:\"

**lpSectorsPerCluster**  
32-bit integer variable which receives the number of sectors in a cluster on the disk.

**lpBytesPerSector**  
32-bit integer variable which receives the number of bytes in a sector on the disk.

**lpNumberOfFreeClusters**  
32-bit integer variable which receives the number of unused, empty clusters on the disk. **Windows 2000:** This may be lower than the actual value if per-user quotas are enabled.

**lpTotalNumberOfClusters**  
32-bit integer variable which receives the total number of clusters, used and unused, on the disk. **Windows 2000:** This may be lower than the actual value if per-user quotas are enabled.

### Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getdiskfreespace.html).

' Calculate and display the free and total space on drive C:
Dim secPerClus As Long  ' receives sectors per cluster
Dim bytePerSec As Long  ' receives bytes per sector
Dim freeClus As Long   ' receives number of free clusters
Dim totalClus As Long   ' receives total number of clusters
Dim retval As Long      ' return value

' Read the information into the variables
retval = GetDiskFreeSpace("c:\", secPerClus, bytePerSec, freeClus, totalClus)

' Display the information
Debug.Print "Free space:"; freeClus * secPerClus * bytePerSec; "bytes"
Debug.Print "Total space:"; totalClus * secPerClus * bytePerSec; "bytes"

### See Also

GetDiskFreeSpaceEx

### Category

Files

[Go back to the alphabetical Function listing.](http://216.26.168.92/vbapi/ref/g/getdiskfreespace.html)

[Go back to the Reference section index.]
Last Modified: July 25, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getdiskfreespace.html
GetDiskFreeSpaceEx Function

Declare Function GetDiskFreeSpaceEx Lib "kernel32.dll" Alias "GetDiskFreeSpaceExA" (ByVal lpDirectoryName As String, lpFreeBytesAvailableToCaller As ULARGE_INTEGER, lpTotalNumberOfBytes As ULARGE_INTEGER, lpTotalNumberOfFreeBytes As ULARGE_INTEGER) As Long

Platforms

- **Windows 95**: Requires OEM Service Release 2 (OSR2) or later.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

GetDiskFreeSpaceEx determines information about the size of a disk. It finds the free space available to the current user, the total disk space, and the amount of free space (all in bytes). Each value is placed into a ULARGE_INTEGER structure which can hold the unsigned 64-bit integer values. (If your programming language supplies an intrinsic unsigned 64-bit integer data type, that can be used instead.)

Return Value

If an error occurred, the function returns 0 (use GetLastError to retrieve the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

*lpDirectoryName*

The name of a directory on the disk to retrieve the size information about. While this can be the name of the disk's root directory, it doesn't have to be.

*lpFreeBytesAvailableToCaller*

Unsigned 64-bit integer variable which receives the amount of free disk space available, in bytes, to the user.

*Windows 2000*:

This may be lower than the actual value if per-user disk space quotas are enabled.
Windows API Guide: GetDiskFreeSpaceEx Function

**lpTotalNumberOfBytes**
Unsigned 64-bit integer variable which receives the amount of total disk space, in bytes. **Windows 2000**: This may be lower than the actual value if per-user disk space quotas are enabled.

**lpTotalNumberOfFreeBytes**
Unsigned 64-bit integer variable which receives the amount of free disk space, in bytes.

**Example**

Display the total free space available on drive C:. Because Visual Basic doesn't have good support for the 64-bit integers needed for modern hard drive's free spaces, the workaround described on this page is needed to display the values properly. This example runs when the user click button Command1, so to use this example, you need to place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type UINTEGER
    LowPart As Long
    HighPart As Long
End Type
Public Declare Function GetDiskFreeSpaceEx Lib "kernel32.dll" Alias "GetDiskFreeSpaceExA" (ByVal _
    lpDirectoryName As String, lpFreeBytesAvailableToCaller As UINTEGER, _
    lpTotalNumberOfBytes As UINTEGER, lpTotalNumberOfFreeBytes As UINTEGER) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

' *** Place the following code inside the form window. ***
Private Sub Command1_Click()
    Dim userbytes As UINTEGER ' bytes free to user
    Dim totalbytes As UINTEGER ' total bytes on disk
    Dim freebytes As UINTEGER ' free bytes on disk
    Dim tempval As Currency ' display buffer for 64-bit values
    Dim retval As Long ' generic return value

    ' Get information about the C: drive.
    retval = GetDiskFreeSpaceEx("C:\", userbytes, totalbytes, freebytes)
    ' Copy freebytes into the Currency data type.
    CopyMemory tempval, freebytes, 8
    ' Multiply by 10,000 to move Visual Basic's decimal point to the end of the actual number.
    tempval = tempval * 10000
    ' Display the amount of free space on C:.
    Debug.Print "Free Space on the C: drive: "; tempval; "bytes"
End Sub
See Also

GetDiskFreeSpace

Category

Files

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getdiskfreespaceex.html
GetDoubleClickTime Function

Declare Function GetDoubleClickTime Lib "user32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

GetDoubleClickTime determines the maximum time allowed between successive mouse clicks for Windows to interpret it as a double click (along with negligible mouse movement). The maximum time between clicks is given in milliseconds. The function returns the maximum double click time.

Example:

' Display the maximum amount of time between clicks to consider the operation a double click.
Dim doubletime As Long ' receives double click time
doubletime = GetDoubleClickTime() ' get the maximum double click time
Debug.Print doubletime; "milliseconds are allowed between clicks during a double click."

See Also: SetDoubleClickTime
Category: Mouse

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetDriveType Function

Declare Function GetDriveType Lib "kernel32.dll" Alias "GetDriveTypeA" (ByVal nDrive As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetDriveType finds the type of disks a disk drive is/uses. This could be a fixed (hard) drive, a floppy drive, a CD-ROM drive, etc. The function returns the drive type. 0 means that an error occurred. 1 means that the specified drive does not exist. Other return values are one of the following flags identifying the drive type:

- DRIVE_CDROM = 5
  - A CD-ROM drive.
- DRIVE_FIXED = 3
  - A hard drive.
- DRIVE_RAMDISK = 6
  - A RAM disk.
- DRIVE_REMOTE = 4
  - A network drive or a drive located on a network server.
- DRIVE_REMOVABLE = 2
  - A floppy drive or some other removable-disk drive.

nDrive
The root directory of the drive to check, such as "c:" or "a:"

Example:

' Determine what type of drive D: is
Dim drivetype As Long ' receives the drive type

drivetype = GetDriveType("d:\") ' determine which kind of drive this is
If drivetype = 1 Then Debug.Print "Drive D:\ does not exist."
If drivetype = DRIVE_REMOVABLE Then Print "Drive D:\ is a removable-disk drive."
If drivetype = DRIVE_FIXED Then Print "Drive D:\ is a hard drive."
If drivetype = DRIVE_CDROM Then Print "Drive D:\ is a CD-ROM drive."
' etc.

Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetEnvironmentVariable Function

Declare Function GetEnvironmentVariable Lib "kernel32.dll" Alias "GetEnvironmentVariableA" (ByVal lpName As String, ByVal lpBuffer As String, ByVal nSize As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

GetEnvironmentVariable reads the value of one of the computer's environment variables. The value is placed into the string buffer passed as the *lpBuffer* parameter.

Return Value

If the specified environment variable does not exist, the function returns zero. If the string passed as *lpBuffer* was too short to receive the environment variable's value, the function returns the necessary minimum buffer length. If successful, the function returns the length of the string copied into *lpBuffer*, not counting the terminating null character.

Visual Basic-Specific Issues

None.

Parameters

- **lpName**
  
  The name of the environment variable to read.

- **lpBuffer**
  
  Receives the value of the environment variable. This string must be sufficiently long to receive the value copied into it.

- **nSize**
  
  The length of the string passed as *lpBuffer*. 
Example

Read the BLASTER environment variable, which contains some information about the Sound Blaster configuration on the computer. Typically, this variable is only used by DOS programs, so it isn't very useful for Windows applications. But then again, this is just an example of how to use GetEnvironmentVariable. The example runs when button Command1 is clicked, so you must place a command button named Command1 on a form window before running this example.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetEnvironmentVariable Lib "kernel32.dll" Alias "GetEnvironmentVariableA" (ByVal _
    lpName As String, ByVal lpBuffer As String, ByVal nSize As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim envvar As String ' receives the value of the environment variable
    Dim slength As Long ' length of the string copied into envvar
    ' Make enough room in envvar to receive the data.
    envvar = Space(256)
    ' Read the value of the BLASTER environment variable.
    slength = GetEnvironmentVariable("BLASTER", envvar, Len(envvar))
    If slength = 0 Then
        Debug.Print "The BLASTER environment variable does not exist on this system."
    Else
        ' Remove the terminating null and everything following it.
        envvar = Left(envvar, slength)
        Debug.Print "BLASTER = "; envvar
    End If
End Sub

See Also

SetEnvironmentVariable

Category

Processes & Threads

Back to the Function list.
Back to the Reference section.
GetFileAttributes Function

Declare Function GetFileAttributes Lib "kernel32.dll" Alias "GetFileAttributesA" (ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetFileAttributes returns the attributes of a file or a directory. Attributes determine such things as read-only status, archive status (most files are), hidden status, etc. If the function fails, it will return 0. If the file or directory cannot be found, it will return -1. Otherwise, the return value will be one or more of the following file attribute flags:

FILE_ATTRIBUTE_ARCHIVE = &H20
  An archive file (which most files are).
FILE_ATTRIBUTE_COMPRESSED = &H800
  A file residing in a compressed drive or directory.
FILE_ATTRIBUTE_DIRECTORY = &H10
  A directory instead of a file.
FILE_ATTRIBUTE_HIDDEN = &H2
  A hidden file, not normally visible to the user.
FILE_ATTRIBUTE_NORMAL = &H80
  An attribute-less file (cannot be combined with other attributes).
FILE_ATTRIBUTE_READONLY = &H1
  A read-only file.
FILE_ATTRIBUTE_SYSTEM = &H4
  A system file, used exclusively by the operating system.

lpFileName
  The full name of the file or directory to check the attributes of, including the full path.

Example:

' Display the attributes of C:\Files\program.exe
Dim attribs As Long  ' receives file attributes
attribs = GetFileAttributes("C:\Files\program.exe")  ' read file attributes
If (attribs And FILE_ATTRIBUTES_ARCHIVE) <> 0 Then Debug.Print "Archive"
If (attribs And FILE_ATTRIBUTES_HIDDEN) <> 0 Then Debug.Print "Hidden"
If (attrs And FILE_ATTRIBUTES_READONLY) <> 0 Then Debug.Print "Read-only"
' etc....

See Also: GetFileInformationByHandle, SetFileAttributes
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getfileattributes.html
GetFileInformationByHandle Function

Declare Function GetFileInformationByHandle Lib "kernel32.dll" (ByVal hFile As Long, lpFileInformation As BY_HANDLE_FILE_INFORMATION) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetFileInformationByHandle determines various information about a file. This information includes the file's attributes; its creation, last-access, and last-modified dates and times; the serial number of the disk the file is on; the size of the file; the number of links to it in the file system; and the its unique numeric identifier. All of this information is put into the structure passed as lpFileInformation. The function returns 0 if an error occurred, or 1 if successful.

**hFile**
A handle to the file to get the information of.

**lpFileInformation**
Receives the information (specified above) relating to the file.

**Example:**

```
' Display the serial number of the disk that file C:\MyProgram\datafile.txt
' is on -- in other words, we are finding the serial number of drive C:. Note that here we
' aren't interested in the other information we receive. Also note that the
' alternate declare
' of CreateFile must be used here since we're using Win 95/98 -- see its page for
details.
Dim hfile As Long  ' receives the handle to the file
Dim fileinfo As BY_HANDLE_FILE_INFORMATION  ' receives info about the file
Dim hexstring As String  ' will receive the hexadecimal form of the serial number
Dim retval As Long  ' return value

' Get a handle to the file (note how the alternate declare is used):
hfile = CreateFileNS("C:\MyProgram\datafile.txt", GENERIC_READ, FILE_SHARE_READ, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then  ' if the file could not be opened
    Debug.Print "Could not open the file C:\MyProgram\datafile.txt."
    End  ' abort the program
End If

' Display the serial number, using hexadecimal:
retval = GetFileInformationByHandle(hfile, fileinfo)  ' read the information
hexstring = Hex(fileinfo.dwVolumeSerialNumber)  ' get the hexadecimal value of the
serial number
```

If Len(hexstring) < 8 Then ' if the string is less than 8 characters,
    hexstring = String("0", 8 - Len(hexstring)) & hexstring ' then right-pad it with "0"s
End If
Debug.Print "The serial number of C: is "; hexstring
' Close the file:
retval = CloseHandle

See Also: GetFileAttributes, GetFileSize, GetFileTime
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetFileSize Function

Declare Function GetFileSize Lib "kernel32.dll" (ByVal hFile As Long, lpFileSizeHigh As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetFileSize determines the size of the file. The file size is given in a 64-bit value that is split into two 32-bit values. The high-order half is put into the variable passed as lpFileSizeHigh; the low-order half is returned by the function. To get the size, you can either put the binary or hexadecimal values of the two variables side-by-side, or use the formula filesize = lpFileSizeHigh * 2^32 + return value. If an error occurs, the function instead returns -1.

hFile
A handle to the file to determine the size of. The file must be opened with at least either read-level or write-level access.

lpFileSizeHigh
Variable that receives the high-order half of the file size.

Example:

' Display the file size of "C:\MyProgram\datafile.txt". Note how
' the alternate declare of the CreateFile function (needed to get the file's handle)
' must be used -- see that function's page for details.
Dim hfile As Long  ' receives a handle to the file
Dim loworder As Long, highorder As Long  ' receive the low- and high-order halves of
the file size
Dim retval As Long  ' return value

' Get a handle to the file using CreateFile's alternate declare (necessary for non-
Win NT).
hfile = CreateFileNS("C:\MyProgram\datafile.txt", GENERIC_READ, FILE_SHARE_READ, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then  ' error opening the file
  Debug.Print "Could not open file C:\MyProgram\datafile.txt"
End  ' abort the program
End If

' Read and display that file's size in bytes.
highorder = 0  ' initialize the value for high-order half
loworder = GetFileSize(hfile, highorder)  ' read the file's size
If highorder = 0 Then  ' if there is no high-order part
  Debug.Print "File size:"; loworder; "bytes"  ' display the file size
Else  ' if there is a high-order part (file size >= 4.29 GB!)
  ' Visual Basic has no 64-bit variables, so we can't display the actual value:
  Debug.Print "File size:"; highorder; "* 2^32 +"; loworder; "bytes (in base-10)"
  ' But we can combine the two hex values to give the result in hexadecimal:
Debug.Print "File size: " & Hex(highorder) & Hex(loworder) & " bytes (in hexadecimal)"

End If

' Close the file
retval = CloseHandle(hfile) ' close the handle

See Also: GetFileInformationByHandle
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetFileTime Function

Declare Function GetFileTime Lib "kernel32.dll" (ByVal hFile As Long, lpCreationTime As FILETIME, lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetFileTime determines the times and dates of creation, last access, and last modification (write-to) of a file. Each of these times are placed in the corresponding structures passed to the function. All times obtained by the function are in UTC time (Coordinated Universal Time, a.k.a. Greenwich Mean Time (GMT)), not in the system's local time. Note that, depending on the operating system, the exact resolution of file times may vary -- the file times obtained by the function may not correspond to the actual date and time of creation/access/modification simply because the operating system does not store the information that precisely.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

hFile
A handle to the file to obtain the creation, last access, and last modification times and dates of. The file must have been opened with at least read-level access.

lpCreationTime
Receives the time and date of when the file was created.

lpLastAccessTime
Receives the time and date of when the file was last accessed.
lpLastWriteTime

Receives the time and date of when the file was last written to or modified.

Example

' This code is licensed according to the terms and conditions listed here.

' Display the date on which the file C:\MyApp\test.txt was
' created. Note how the time zone conversion is necessary.
Dim hFile As Long  ' handle to the opened file
Dim ctime As FILETIME  ' receives time of creation
Dim atime As FILETIME  ' receives time of last access
Dim mtime As FILETIME  ' receives time of last modification
Dim thetime As SYSTEMTIME  ' used to manipulate the time
Dim retval As Long  ' return value

' First, open the file C:\MyApp\test.txt for read-level access. Note the
' expression necessary to pass 0 as lpSecurityAttributes.
hFile = CreateFile("C:\MyApp\test.txt", GENERIC_READ, FILE_SHARE_READ, ByVal CLng(0),
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hFile = -1 Then
    Debug.Print "Could not open the file successfully -- aborting."
    End  ' terminate the program
End If

' Next, get the creation, last-access, and last-modification times.
retval = GetFileTime(hFile, ctime, atime, mtime)
' Convert the creation time to the local time zone.
retval = FileTimeToLocalFileTime(ctime, ctime)
' Convert the FILETIME format to the SYSTEMTIME format.
retval = FileTimeToSystemTime(ctime, thetime)

' Display the date of creation of the file to the user.
Debug.Print "The file was created on "; thetime.wMonth; "-"; thetime.wDay; "-";
thetime.wYear

' Close the file to free up resources.
retval = CloseHandle(hFile)

See Also

GetFileInformationByHandle, SetFileTime

Category

Files

Go back to the alphabetical Function listing.
GetFileVersionInfo Function

Declare Function GetFileVersionInfo Lib "version.dll" Alias "GetFileVersionInfoA" (ByVal lptstrFilename As String, ByVal dwHandle As Long, ByVal dwLen As Long, lpData As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetFileVersionInfo extracts the version information resource stored inside a 32-bit executable-type file. The version information resource is a series of bytes that is difficult to read directly. Use the VerQueryValue to retrieve the desired pieces of data from the resource.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- lptstrFilename
  - The full path and filename of the file.
- dwHandle
  - Reserved -- set to 0.
- dwLen
  - The size in bytes of the buffer passed as lpData.
- lpData
  - A buffer, such as a byte array, that receives the version information resource of the file.
Example

Display information about the file whose full path and filename is entered into textbox Text1. Display the version number, copyright information, and file description when button Command1 is pressed. To use this example, you obviously have to enter the filename of a 32-bit executable/DLL/etc. into Text1. Obviously, this example requires that you create a text box called Text1 and a command button called Command1.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
(ByVal lptstrFilename As String, ByVal dwHandle As Long, ByVal dwLen As Long,
(lpData As Any) As Long
Public Declare Function GetFileVersionInfoSize Lib "version.dll" Alias _
"GetFileVersionInfoSizeA" (ByVal lptstrFilename As String, lpdwHandle As Long) As Long
Public Declare Function VerQueryValue Lib "version.dll" Alias "VerQueryValueA" (pBlock _
As Any, ByVal lpSubBlock As String, lplpBuffer As Long, puLen As Long) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 _
As Any, ByVal lpString2 As Any) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, _
Source As Any, ByVal Length As Long)
Public Type VS_FIXEDFILEINFO
    dwSignature As Long
    dwStrucVersion As Long
    dwFileVersionMS As Long
    dwFileVersionLS As Long
    dwProductVersionMS As Long
    dwProductVersionLS As Long
    dwFileFlagsMask As Long
    dwFileFlags As Long
    dwFileOS As Long
    dwFileType As Long
    dwFileSubtype As Long
    dwFileDateMS As Long
End Type
Public Const VFT_APP = &H1
Public Const VFT_DLL = &H2
Public Const VFT_DRV = &H3
Public Const VFT_VXD = &H5

' *** Place the following function definitions inside a module. ***
HIWORD and LOWORD are API macros defined below.

Public Function HIWORD (ByVal dwValue As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(dwValue), 8)
    HIWORD = CLng("&H" & Left(hexstr, 4))
End Function

Public Function LOWORD (ByVal dwValue As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(dwValue), 8)
    LOWORD = CLng("&H" & Right(hexstr, 4))
End Function

This nifty subroutine swaps two byte values without needing a buffer variable.
This technique, which uses Xor, works as long as the two values to be swapped are numeric and of the same data type (here, both Byte).

Public Sub SwapByte (byte1 As Byte, byte2 As Byte)
    byte1 = byte1 Xor byte2
    byte2 = byte1 Xor byte2
    byte1 = byte1 Xor byte2
End Sub

This function creates a hexadecimal string to represent a number, but it outputs a string of a fixed number of digits. Extra zeros are added to make the string the proper length. The "&H" prefix is not put into the string.

Public Function FixedHex (ByVal hexval As Long, ByVal nDigits As Long) As String
    FixedHex = Right("00000000" & Hex(hexval), nDigits)
End Function

*** Place the following code inside the form that has Command1 and Text1. ***

Private Sub Command1_Click()
    Dim vffi As VS_FIXEDFILEINFO  ' version info structure
    Dim buffer() As Byte          ' buffer for version info resource
    Dim pData As Long             ' pointer to version info data
    Dim nDataLen As Long          ' length of info pointed at by pData
    Dim cpl(0 To 3) As Byte       ' buffer for code page & language
    Dim cplstr As String          ' 8-digit hex string of cpl
    Dim dispstr As String         ' string used to display version information
    Dim retval As Long            ' generic return value

    ' First, get the size of the version info resource. If this function fails, then Text1 identifies a file that isn't a 32-bit executable/DLL/etc.
    nDataLen = GetFileVersionInfoSize(Text1.Text, pData)
    If nDataLen = 0 Then
        Debug.Print "Not a 32-bit executable!"
        Exit Sub
    End If

    ' Make the buffer large enough to hold the version info resource.
    ReDim buffer(0 To nDataLen - 1) As Byte
    ' Get the version information resource.
    retval = GetFileVersionInfo(Text1.Text, 0, nDataLen, buffer(0))
Get a pointer to a structure that holds a bunch of data.

```
retval = VerQueryValue(buffer(0), ",", pData, nDataLen)
```

Copy that structure into the one we can access.

```
CopyMemory vffi, ByVal pData, nDataLen
```

Display the full version number of the file.

```
dispstr = Trim(Str(HIWORD(vffi.dwFileVersionMS))) & "." & _
            Trim(Str(LOWORD(vffi.dwFileVersionMS))) & "." & _
            Trim(Str(HIWORD(vffi.dwFileVersionLS))) & "." & _
            Trim(Str(LOWORD(vffi.dwFileVersionLS)))
```

Debug.Print "Version Number: "; dispstr

Display the type of file it is (i.e., executable, DLL, etc.).

```
Select Case vffi.dwFileType
        Case VFT_APP
dispstr = "Application"
        Case VFT_DLL
dispstr = "Dynamic Link Library (DLL)"
        Case VFT_DRV
dispstr = "Device Driver"
        Case VFT_VXD
dispstr = "Virtual Device Driver"
        Case Else
dispstr = "Unknown"
End Select
```

Debug.Print "File Type: "; dispstr

Before reading any strings out of the resource, we must first determine the code page and language. The code to get this information follows.

```
retval = VerQueryValue(buffer(0), "\VarFileInfo\Translation", pData, nDataLen)
```

Copy that information into the byte array.

```
CopyMemory cpl(0), ByVal pData, 4
```

It is necessary to swap the first two bytes, as well as the last two bytes.

```
SwapByte cpl(0), cpl(1)
```

```
SwapByte cpl(2), cpl(3)
```

Convert those four bytes into a 8-digit hexadecimal string.

```
cplstr = FixedHex(cpl(0), 2) & FixedHex(cpl(1), 2) & FixedHex(cpl(2), 2) & _
        FixedHex(cpl(3), 2)
```

Cplstr now represents the code page and language to read strings as.

```
retval = VerQueryValue(buffer(0), "\StringFileInfo\" & cplstr & "\LegalCopyright", _,
pData, nDataLen)
```

Copy that data into a string for display.

```
dispstr = Space(nDataLen)
```

```
retval = lstrcpy(dispstr, pData)
```

Display the result.

```
Debug.Print "Copyright Info: "; dispstr
```

Similarly, read a description of the file and display it.
 See Also

GetFileVersionInfoSize, VerQueryValue

Category

Files

Go back to the Function list. 
Go back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getfileversioninfo.html
GetFileVersionInfoSize Function

Declare Function GetFileVersionInfoSize Lib "version.dll" Alias "GetFileVersionInfoSizeA" (ByVal lpstrFilename As String, lpdwHandle As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetFileVersionInfoSize determines the length of the version information resource stored with a 32-bit executable-type file. The actual version information resource is obtained by using GetFileVersionInfo. This function does not work with 16-bit executable-type files, nor with files that are not executable at all.

Return Value

If successful, the function returns the size in bytes of the file's version information resource. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- lpstrFilename
  - The full path and filename of the file.
- lpdwHandle
  - Receives a value of 0. Although this parameter is effectively reserved, you must pass a variable to receive this meaningless value.

Example
Display information about the file whose full path and filename is entered into textbox Text1. Display the version number, copyright information, and file description when button Command1 is pressed. To use this example, you obviously have to enter the filename of a 32-bit executable/DLL/etc. into Text1. Obviously, this example requires that you create a text box called Text1 and a command button called Command1.

This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function GetFileVersionInfo Lib "version.dll" Alias "GetFileVersionInfoA" (ByVal lptstrFilename As String, ByVal dwHandle As Long, ByVal dwLen As Long, lpData As Any) As Long
Public Declare Function GetFileVersionInfoSize Lib "version.dll" Alias "GetFileVersionInfoSizeA" (ByVal lptstrFilename As String, lpdwHandle As Long) As Long
Public Declare Function VerQueryValue Lib "version.dll" Alias "VerQueryValueA" (pBlock As Any, ByVal lpSubBlock As String, lplpBuffer As Long, puLen As Long) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Type VS_FIXEDFILEINFO
    dwSignature As Long
    dwStrucVersion As Long
    dwFileVersionMS As Long
    dwFileVersionLS As Long
    dwProductVersionMS As Long
    dwProductVersionLS As Long
    dwFileFlagsMask As Long
    dwFileFlags As Long
    dwFileOS As Long
    dwFileType As Long
    dwFileSubtype As Long
    dwFileDateMS As Long
    dwFileDateLS As Long
End Type
Public Const VFT_APP = &H1
Public Const VFT_DLL = &H2
Public Const VFT_DRV = &H3
Public Const VFT_VXD = &H5

*** Place the following function definitions inside a module. ***

HIWORD and LOWORD are API macros defined below.
Public Function HIWORD (ByVal dwValue As Long) As Long
    Dim hexstr As String

hexstr = Right("00000000" & Hex(dwValue), 8)
HIWORD = CLng("&H" & Left(hexstr, 4))

End Function

Public Function LOWORD (ByVal dwValue As Long) As Long
Dim hexstr As String
hexstr = Right("00000000" & Hex(dwValue), 8)
LOWORD = CLng("&H" & Right(hexstr, 4))
End Function

' This nifty subroutine swaps two byte values without needing a buffer variable.
' This technique, which uses Xor, works as long as the two values to be swapped are
' numeric and of the same data type (here, both Byte).
Public Sub SwapByte (byte1 As Byte, byte2 As Byte)
byte1 = byte1 Xor byte2
byte2 = byte1 Xor byte2
byte1 = byte1 Xor byte2
End Sub

' This function creates a hexadecimal string to represent a number, but it
' outputs a string of a fixed number of digits. Extra zeros are added to make
' the string the proper length. The "&H" prefix is not put into the string.
Public Function FixedHex (ByVal hexval As Long, ByVal nDigits As Long) As String
FixedHex = Right("00000000" & Hex(hexval), nDigits)
End Function

' *** Place the following code inside the form that has Command1 and Text1. ***

Private Sub Command1_Click()
Dim vffi As VS_FIXEDFILEINFO  ' version info structure
Dim buffer() As Byte          ' buffer for version info resource
Dim pData As Long             ' pointer to version info data
Dim nDataLen As Long          ' length of info pointed at by pData
Dim cpl(0 To 3) As Byte       ' buffer for code page & language
Dim cplstr As String          ' 8-digit hex string of cpl
Dim dispstr As String         ' string used to display version information
Dim retval As Long            ' generic return value

' First, get the size of the version info resource. If this function fails,
then Text1
' identifies a file that isn't a 32-bit executable/DLL/etc.
nDataLen = GetFileVersionInfoSize(Text1.Text, pData)
If nDataLen = 0 Then
    Debug.Print "Not a 32-bit executable!"
    Exit Sub
End If
' Make the buffer large enough to hold the version info resource.
ReDim buffer(0 To nDataLen - 1) As Byte
' Get the version information resource.
retval = GetFileVersionInfo(Text1.Text, 0, nDataLen, buffer(0))

' Get a pointer to a structure that holds a bunch of data.
retval = VerQueryValue(buffer(0), ",\", pData, nDataLen)
' Copy that structure into the one we can access.
CopyMemory vffi, ByVal pData, nDataLen
' Display the full version number of the file.
dispstr = Trim(Str(HIWORD(vffi.dwFileVersionMS))) & "." & _
    Trim(Str(LOWORD(vffi.dwFileVersionMS))) & "." & _
    Trim(Str(HIWORD(vffi.dwFileVersionLS))) & "." & _
    Trim(Str(LOWORD(vffi.dwFileVersionLS)))
Debug.Print "Version Number: "; dispstr
' Display the type of file it is (i.e., executable, DLL, etc.).
Select Case vffi.dwFileType
    Case VFT_APP
        dispstr = "Application"
    Case VFT_DLL
        dispstr = "Dynamic Link Library (DLL)"
    Case VFT_DRV
        dispstr = "Device Driver"
    Case VFT_VXD
        dispstr = "Virtual Device Driver"
    Case Else
        dispstr = "Unknown"
End Select
Debug.Print "File Type: "; dispstr

' Before reading any strings out of the resource, we must first determine the
code page
' and language. The code to get this information follows.
retval = VerQueryValue(buffer(0), "\VarFileInfo\Translation", pData, _
nDataLen)
' Copy that information into the byte array.
CopyMemory cpl(0), ByVal pData, 4
' It is necessary to swap the first two bytes, as well as the last two bytes.
SwapByte cpl(0), cpl(1)
SwapByte cpl(2), cpl(3)
' Convert those four bytes into a 8-digit hexadecimal string.
cplstr = FixedHex(cpl(0), 2) & FixedHex(cpl(1), 2) & FixedHex(cpl(2), 2) & _
    FixedHex(cpl(3), 2)
' cplstr now represents the code page and language to read strings as.

' Read the copyright information from the version info resource.
retval = VerQueryValue(buffer(0), "\StringFileInfo\" & cplstr & _
    "\LegalCopyright\", _
    pData, nDataLen)
' Copy that data into a string for display.
dispstr = Space(nDataLen)
    retval = lstrcpy(dispstr, pData)
' Display the result.
Debug.Print "Copyright Info: "; dispstr
' Similarly, read a description of the file and display it.
retval = VerQueryValue(buffer(0), "\StringFileInfo\" & cplstr & _
    "\FileDescription\", _
    pData, nDataLen)
dispstr = Space(nDataLen)
retval = lstrcpy(dispstr, pData)
Debug.Print "File Description: "; dispstr
End Sub

See Also

GetFileVersionInfo

Category

Files

Back to the Function list.
Back to the Reference section.
GetFocus Function

Declare Function GetFocus Lib "user32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetFocus obtains a handle to the window in the calling thread that has the input focus. If another program has the input focus, however, this function will not work and will report an error.

Return Value

If successful, the function returns a handle to the window that has the input focus. If an error occurred, or if another program has the input focus, the function returns zero (use GetLastErr to get the error code).

Visual Basic-Specific Issues

None.

Parameters

None.

Example

Print the title of the window that currently has the input focus, if our program has it. If not, inform the user that some other program has input focus. This is done whenever timer timCheck elapses, so to use this example, you must first place a timer named timCheck on a form window.

' This code is licensed according to the terms and conditions listed here.
Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function GetFocus Lib "user32.dll" () As Long
Public Declare Function GetWindowTextLength Lib "user32.dll" Alias "GetWindowTextLengthA" 
   (ByVal hWnd As Long) As Long
Public Declare Function GetWindowText Lib "user32.dll" Alias "GetWindowTextA" (ByVal hWnd As Long, _
   ByVal lpString As String, ByVal nMaxCount As Long) As Long

*** Place the following code inside the form window. ***

Private Sub timCheck_Timer()
   Dim hWnd As Long       ' window that has the focus
   Dim wintext As String  ' title of the window
   Dim textlen As Long    ' length of the title
   
   ' First, see which window in the program, if any, has the input focus.
   hWnd = GetFocus()
   If hWnd = 0 Then
      Debug.Print "This program does not have the input focus."
   Else
      ' Get the title of the window that has the focus.
      textlen = GetWindowTextLength(hWnd) + 1
      wintext = Space(textlen)
      textlen = GetWindowText(hWnd, wintext, textlen)
      ' Display the title, removing the terminating null.
      Debug.Print "The window titled " & Left(wintext, textlen) & " has the focus."
   End If
End Sub

See Also

SetFocus

Category

Windows

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
GetForegroundWindow Function

Declare Function GetForegroundWindow Lib "user32.dll" () As Long

Platforms: Win 95/98, Win NT

GetForegroundWindow finds which window is currently the foreground window. The foreground window is the window, usually at the top of the Z-order, with which the user is currently working with -- i.e., the window with the focus. The function returns 0 if an error occurred, or the handle of the foreground window if successful.

Example:

' Display the title bar text of the foreground window.
Dim hforewnd As Long  ' receives handle of foreground window
Dim slength As Long  ' length of foreground window's title bar text
Dim wintext As String  ' buffer for foreground window's title bar text
Dim retval As Long  ' return value

hforewnd = GetForegroundWindow()  ' determine the foreground window
slength = GetWindowTextLength(hforewnd) + 1  ' length of its title bar text
wintext = Space(slength)  ' make room in the buffer to receive the text
retval = GetWindowText(hforewnd, wintext, slength)  ' get title bar text
wintext = Left(wintext, slength - 1)  ' remove null character from end of string
Debug.Print "The window "; wintext; " is the foreground window."

See Also: SetForegroundWindow
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetFullPathName Function

Declare Function GetFullPathName Lib "kernel32.dll" Alias "GetFullPathNameA" (ByVal lpFileName As String, ByVal nBufferLength As Long, ByVal lpBuffer As String, ByVal lpFilePart As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetFullPathName appends a specified filename to the name of the current directory. For example, if you specify the file "hello.txt" and the current directory is "C:\My Documents\Junk", the resulting filename would be "C:\My Documents\Junk\hello.txt". This string is put into the string passed as lpBuffer. The function returns 0 if an error occurred, or the length of the final string if successful.

lpFileName
   The name of the file to append.

nBufferLength
   The size in characters of lpBuffer.

lpBuffer
   A string variabled that receives the null-terminated combined path and filename.

lpFilePart
   ??? (appears to have no effect)

Example:

' Append the filename datafile.dat to C:\Programs\Test
Dim buffer As String  ' receives path and filename string
Dim numchar As Long  ' receives length of buffer after function call
ChDir "\Programs\Test"  ' change current directory to C:\Programs\Test
buffer = Space(255)  ' make room for buffer to receive the string
numchar = GetFullPathName("datafile.dat", 255, buffer, ")  ' put the result string into buffer
buffer = Left(buffer, numchar)  ' extract data from the returned string
Debug.Print buffer  ' display resulting string

See Also: GetShortPathName
Category: Files
gethostbyaddr Function

Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

gethostbyaddr gets information about a host computer. The computer is identified by its network address (usually but not necessary an IP address). The information about the host is placed into a HOSTENT structure, a pointer to which is returned by the function.

Return Value

If successful, the function returns a pointer to a HOSTENT structure that contains data about the host. This pointer is only guaranteed to be valid until the next call to a Winsock function. If an error occurred, the function returns zero (use WSAGetLastError to get the error code.

Visual Basic-Specific Issues

To access the returned data, use CopyMemory to copy the data referenced by the pointer into a HOSTENT structure allocated by your program. See the example for a demonstration of this.

Parameters

- addr
  The address of the host computer to get information about. This address must be in network byte order.
- length
  The length in bytes of the address. For IP addresses, this will be 4.
- protocol
  One of the following flags specifying the type of address specified in addr. The flag identifies the protocol in which the address is used.
  - AF_12844
IEEE 1284.4 WG AF Protocol.

AF_APPLE TALK
Appletalk protocol.

AF_ATM
Native ATM services protocol.

AF_BAN
Banyan protocol.

AF_CCITT
One of the CCITT protocols (such as X.25).

AF_CHAOS
One of the MIT CHAOS protocols.

AF_CLUSTER
Microsoft Wolfpack protocol.

AF_DATAKIT
One of the Datakit protocols.

AF_DECnet
DECnet protocol.

AF_DLI
Direct Data Link interface.

AF_ECMA
A European Computer Manufacturers protocol.

AF_FIREFOX
A FireFox protocol.

AF_HYLINK
NSC Hyperchannel protocol.

AF_IMPLINK
Arpanet IMP address.

AF_INET
Internet or other inter-network address (such as UDP/IP or TCP/IP).

AF_INET6
Internet or other inter-network address using IPv6 addresses.

AF_IPX
One of the IPX protocols (such as IPX or SPX).

AF_ISO
One of the ISO protocols.

AF_LAT
LAT protocol.

AF_NETBIOS
NetBIOS protocol.

AF_NS
One of the Xerox NS protocols, including IPX.

AF_OSI
One of the ISO protocols. (Same as AF_ISO.)

AF_PUP
A PUP protocol address.

AF_SNA
IBM SNA protocol.

AF_UNIX
A Unix-type local-to-host pipe or portal.

AF_UNKNOWN1
An unknown protocol.

AF_VOICEVIEW
Constant Definitions

Const AF_12844 = 25
Const AF_APPLETALK = 16
Const AF_ATM = 22
Const AF_BAN = 21
Const AF_CCITT = 10
Const AF_CHAOS = 5
Const AF_CLUSTER = 24
Const AF_DATAKIT = 9
Const AF_DECnet = 12
Const AF_DLI = 13
Const AF_ECMA = 8
Const AF_FIREFOX = 19
Const AF_HYLINK = 15
Const AF_IMPLINK = 3
Const AF_INET = 2
Const AF_INET6 = 23
Const AF_IPX = 6
Const AF_ISO = 7
Const AF_LAT = 14
Const AF_NETBIOS = 17
Const AF_NS = 6
Const AF_OSI = 7
Const AF_PUP = 4
Const AF_SNA = 11
Const AF_UNIX = 1
Const AF_UNKNOWN1 = 20
Const AF_VOICEVIEW = 18

Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
iMaxUdpDg As Long
lpVendorInfo As Long

End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData _
As WSAData) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As _ INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_ISBLANK = &H469
Public Const IPM_GETADDRESS = &H466
' *** Place the following code in a form window. ***

Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the
' upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to
        register
    Dim retval As Long ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
        Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the
' address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long ' the IP address, in host byte order
    Dim ipAddress_n As Long ' the IP address, in network byte order
    Dim sockinfo As WSADATA ' information about the Winsock implementation
    Dim pHostinfo As Long ' pointer to information about the host computer
    Dim hostinfo As HOSTENT ' information about the host computer
    Dim domainName As String ' the primary domain name of the host computer
    Dim retval As Long ' generic return value

    ' Verify that an IP address was entered.
    retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
    If retval <> 0 Then
        Debug.Print "No IP address was entered!"
        Exit Sub
    End If

    ' Get the IP address entered by the user and verify that all
    ' four fields in the address were entered.
    retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
If retval < 4 Then
    Debug.Print "An incomplete IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)
' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also

gethostbyname

Category

Winsock

Copyright Information Revised October 29, 2000
Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/gethostbyaddr.html
gethostbyname Function

Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

gethostbyname gets information about a host computer. The computer is identified by its domain name. The information about the host is placed into a HOSTENT structure, a pointer to which is returned by the function. Note that gethostbyname cannot identify a host computer by its IP address string. (To get information about a computer with a particular IP address, use gethostbyaddr instead.)

Return Value

If successful, the function returns a pointer to a HOSTENT structure that contains data about the host. This pointer is only guaranteed to be valid until the next call to a Winsock function. If an error occurred, the function returns zero (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

To access the returned data, use CopyMemory to copy the data referenced by the pointer into a HOSTENT structure allocated by your program. See the example for a demonstration of this.

Parameters

*name*

The domain name of the network host to get information about. This cannot be an IP address string.

Example

Print the IP address associated with a domain name specified by the user. Winsock is briefly used to resolve the domain name.
and format a printable IP address string. The user enters the domain name to resolve in text box txtDomain, and the domain name is resolved when the user clicks button cmdGetIP.

To run this example, place a text box named txtDomain and a command button named cmdGetIP on a form window.

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/gethostbyname.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type **WSADATA**
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function **WSAStartup** Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSADatas As **WSADATA**) As Long
Public Declare Function **WSACleanup** Lib "wsock32.dll" () As Long

Public Type **HOSTENT**
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function **gethostbyname** Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function **inet_ntoa** Lib "wsock32.dll" (ByVal inaddr As Long) As Long
Public Declare Sub **CopyMemory** Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

' Define a relevant API macro.
Public Function **MAKEWORD** (ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val(&H & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***
Private Sub cmdGetIP_Click()
    Dim sockinfo As WSADATA ' information about Winsock
    Dim hostinfo As HOSTENT ' information about an Internet host
    Dim pHostinfo As Long    ' pointer to a HOSTENT structure
    Dim IPAddress As Long    ' an IP address, packed into a dword
    Dim pIPString As Long    ' pointer to an IP address formatted as a string
    Dim ipAddress As Long    ' an IP address, packed into a dword
    Dim ipString As String   ' holds a human-readable IP address string
    Dim retval As Long       ' generic return value

    ' Open up a Winsock session, using version 2.2.
    retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
    If retval <> 0 Then
        Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval;
        Exit Sub
    End If

    ' Get information about the domain specified in txtDomain.
    pHostinfo = gethostbyname(txtDomain.Text)
    If pHostinfo = 0 Then
        Debug.Print "Unable to resolve domain name."
    Else
        ' Copy the data into a HOSTENT structure.
        CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
        If hostinfo.h_addrtype <> AF_INET Then
            Debug.Print "A non-IP address was returned."
        Else
            ' Copy the pointer to the first (and probably only) IP
            ' address in the structure.
            CopyMemory IPAddress, ByVal hostinfo.h_addr_list, 4
            ' Copy the actual IP address.
            CopyMemory ipAddress, ByVal IPAddress, 4
            ' Convert the IP address into a human-readable string.
            pIPString = inet_ntoa(ipAddress)
            ' Copy the result into a string variable.
            ipString = Space(lstrlen(pIPString))
            retrval = lstrcpy(ipString, pIPString)
            ' Print the result: a human-readable IP address.
            Debug.Print ipString
        End If
    End If

    ' Close the Winsock session.
    retval = WSACleanup()
End Sub

See Also

gethostbyaddr
GetKeyboardState Function

Declare Function GetKeyboardState Lib "user32.dll" (lpKeyState As Byte) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

GetKeyboardState retrieves the state of every key on the keyboard and places the information into an array. Each element of the 256-element array identifies information about the virtual-key whose virtual-key code matches the index of the element. If the &H1 bit is set, that key is toggled. If the &H80 bit is set, the key is currently pressed down. The keyboard information retrieved by this function is thread-specific; its information does not necessarily reflect key states pertaining to the system as a whole.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

*lpKeyState*
A 256-element byte array which receives the key status information for all virtual-keys. Each key is identified by the element corresponding with the key's virtual key code. **Windows NT, 2000:** In addition to the virtual keys, the array also receives information which distinguish between the left and right Ctrl, Alt, and Shift keys, which are placed in the array at the following indices:

- **VK_LSHIFT**
  - The left Shift key.
- **VK_RSHIFT**
  - The right Shift key.
- **VK_LCONTROL**
  - The left Ctrl key.
- **VK_RCONTROL**
  - The right Ctrl key.
- **VK_LMENU**
  - The left Alt key.
- **VK_RMENU**
  - The right Alt key.

### Constant Definitions

```
Const VK_LSHIFT = &HA0
Const VK_RSHIFT = &HA1
Const VK_LCONTROL = &HA2
Const VK_RCONTROL = &HA3
Const VK_LMENU = &HA4
Const VK_RMENU = &HA5
```

### Example

```
' This code is licensed according to the terms and conditions listed here.

' Set the toggle status for every key on the keyboard to "not toggled." This change only applies to the current thread.
Dim keystates(0 To 255) As Byte  ' holds states of entire keyboard
Dim c As Integer  ' counter variable
Dim retval As Long  ' return value

' First, get the current state of the keyboard.
retval = GetKeyboardState(keystates(0))

' Now, loop through each element and explicitly set the toggle bit to 0.
For c = 0 To 255
  ' Make sure the &H1 bit is not set.
  keystates(c) = keystates(c) And (Not &H1)
Next c
```
' Finally, set this to the current keyboard state.
retval = SetKeyboardState(keystates(0))

See Also
GetKeyState, SetKeyboardState

Category
Keyboard

Go back to the alphabetical Function listing.
Go back to the Reference section index.
gethostname Function

Declare Function gethostname Lib "wsock32.dll" (ByVal name As String, ByVal namelen As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

gethostname retrieves the network host name for the computer. This could be a simple host name or a fully-qualified domain name. No matter what it is, though, it can be successfully given to gethostbyname to get information about the computer as a network host (for example, to get its IP address).

Return Value

If successful, the function returns zero. If an error occurred, the function returns a non-zero value (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

name

String that receives the host name of the computer, terminated by a null character. This string must have sufficient room to receive the string copied into it.

namelen

The length of the string passed as name.

Example

When the form window opens, create an IP address control in the upper-left corner and initialize it to the IP address of the
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type

Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long

Public Declare Function WSACleanup Lib "wsock32.dll" () As Long

Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type

Public Const AF_INET = 2

Public Declare Function gethostname Lib "wsock32.dll" (ByVal name As String, ByVal namelen As Long) As Long

Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long

Public Declare Function ntohl Lib "wsock32.dll" (ByVal hostlong As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)

Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long

Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type

Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long

Public Const ICC_INTERNET_CLASSES = &H800

Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal http://216.26.168.92/vbapi/ref/g/gethostname.html (2 of 5) [9/1/2002 5:24:30 PM]
ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hWnd As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long

Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_SETADDRESS = &H465

' Define a relevant API macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***
Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
Dim sockinfo As WSADATA ' information about Winsock
Dim hostinfo As HOSTENT ' information about an Internet host
Dim pHostinfo As Long ' pointer to a HOSTENT structure
Dim localhostName As String ' the computer's domain name
Dim pIPAddress As Long ' pointer to an IP address dword
Dim ipAddress_n As Long ' the IP address in network byte order
Dim ipAddress_h As Long ' the IP address in host byte order
Dim retval As Long ' generic return value

' Register the IP Address control window class.
With comctls
    .dwSize = Len(comctls)
    .dwICC = ICC_INTERNET_CLASSES
End With
retval = InitCommonControlsEx(comctls)

' Create the IP Address control in the corner of the window.
hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
    Me.hWnd, 0, App.hInstance, ByVal CLng(0))

' Open a new Winsock session (version 2.2).
retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Get the domain name of the computer, or, failing that, a string
' that gethostbyname can handle to give an IP address.
localhostName = Space(256)
retval = gethostname(localhostName, 256)
localhostName = Left(localhostName, InStr(localhostName, vbNullChar) - 1)
' Get information about this computer on the network.
' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(localhostName)
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve domain name."
Else
    ' Copy the data into a HOSTENT structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print "A non-IP address was returned."
    Else
        ' Copy the pointer to the first (and probably only) IP
        address in the structure.
        CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
        ' Copy the actual IP address.
        CopyMemory ipAddress_n, ByVal pIPAddress, 4
        ' Convert it to host byte order.
        ipAddress_h = ntohl(ipAddress_n)
        ' Set the IP Address control to hold this address.
        retval = SendMessage(hIPControl, IPM_SETADDRESS, _
            ByVal CLng(0), ByVal ipAddress_h)
    End If
End If
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

Category
Winsock

Back to the Function list.
Back to the Reference section.
GetKeyState Function

Declare Function GetKeyState Lib "user32.dll" (ByVal nVirtKey As Long) As Integer

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetKeyState determines the current status of a key. The function both finds whether the key is currently pressed down or not, and determines if the key is currently toggled. The keyboard information retrieved by this function is thread-specific; its information does not necessarily reflect key states pertaining to the system as a whole.

Return Value

If the &H1 bit of the return value is set, the key is toggled. If the &H8000 bit of the return value is set, the key is currently pressed down.

Visual Basic-Specific Issues

None.

Parameters

nVirtKey
The virtual-key code of the key to read the status of. Windows NT, 2000: This could also be one of the following flags which distinguish between the left and right Ctrl, Alt, and Shift keys:
- VK_LSHIFT
  The left Shift key.
- VK_RSHIFT
  The right Shift key.
VK_LCONTROL
   The left Ctrl key.

VK_RCONTROL
   The right Ctrl key.

VK_LMENU
   The left Alt key.

VK_RMENU
   The right Alt key.

**Constant Definitions**

Const VK_LSHIFT = &HA0
Const VK_RSHIFT = &HA1
Const VK_LCONTROL = &HA2
Const VK_RCONTROL = &HA3
Const VK_LMENU = &HA4
Const VK_RMENU = &HA5

**Example**

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getkeystate.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetKeyState Lib "user32.dll" (ByVal nVirtKey As Long) As Integer

' Determine whether the Q key is currently being pressed.
' The code runs when button Command1 is pressed.

Private Sub Command1_Click()
   Dim keystate As Integer  ' state of the Q key

   ' Get the state of the Q key as returned by the function.
   ' (vbKeyQ is a VB-defined constant for Q's virtual-key code)
   keystate = GetKeyState(vbKeyQ)

   ' Check the &H8000 bit of the return value.
   If keystate And &H8000 Then
      Debug.Print "The Q key is currently down."
   Else
      Debug.Print "The Q key is currently up."
   End If

End Sub

**See Also**

GetAsyncKeyState, GetKeyboardState
Category

Keyboard

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetLastError Function

Declare Function GetLastError Lib "kernel32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetLastError obtains the error code returned by the last API function called. Most API functions merely return a number saying if an error occured, but not specifying exactly which error occured. This function gets a universal error code identifying the type of error that last occured. Note that most functions set the code to 0 (success) if the function completes successfully, erasing the previous error code. Therefore, be sure to check this error code immediately after an error is found.

Return Value

The function returns the error code of the last API function called by the program.

Visual Basic-Specific Issues

Although GetLastError works perfectly in Visual Basic, it will sometimes not appear to work. This is because Visual Basic implicitly uses the API frequently to perform tasks which are seemingly intrinsic to the language. These hidden API function calls will usually overwrite the error code which your code may be trying to read. To compensate for this, the LastDllError property of the Err object, predefined in Visual Basic, caches the error code from the last API function explicitly called by your program. You should use the expression Err.LastDllError instead of the GetLastError function to debug failed API function calls.

Parameters

None.

Example
' This code is licensed according to the terms and conditions listed here.

' Demonstrate catching an invalid handle error.
' If this code does not work, try replacing "GetLastError()" with Err.LastDllError.
Dim retval As Long  ' return value of function
Dim errorcode As Long  ' error code

' Make an invalid call to the following function by giving it an invalid handle
retval = CloseHandle(-1)  ' there is no handle -1!
If retval = 0 Then  ' the return value will be 0 if an error occured
    errorcode = GetLastError()  ' find the error code
    If errorcode = 6 Then Debug.Print "ERROR: Invalid Handle Specified"  ' error 6 = invalid handle
End If

See Also

CommDlgExtendedError, SetLastError, SetLastErrorEx

Category

Errors

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: January 5, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getlasterror.html
GetLocalTime Function

Declare Sub GetLocalTime Lib "kernel32.dll" (lpSystemTime As SYSTEMTIME)

Platforms: Win 95/98, Win NT

GetLocalTime returns the system's time and date. The various features of the time and date (month, day, hour, minute, second, etc.), down to the millisecond, are sorted in the variable passed as lpSystemTime! Windows considers "local time" to be the system's current time.

lpSystemTime
Variable that receives the computer's date and time.

Example:

' Print the current date in mm-dd-yyyy format
Dim localtime As SYSTEMTIME  ' receives the computer's time and date
GetLocalTime localtime  ' read the computer's time and date
' Display the date
Debug.Print localtime.wMonth; "-"; localtime.wDay; "-"; localtime.wYear

See Also: GetSystemTime
Category: Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
GetLogicalDrives Function

Declare Function GetLogicalDrives Lib "kernel32.dll" () As Long

Platforms: Win 32s, Win 95/98, Win NT

GetLogicalDrives determines all the valid logical drives on the computer. Logical drives are any drives assigned a one-letter name (such as A: or C:). The return value is a collection of single-bit flags identifying the drives found. Perform a binary And between the return value and increasing powers of 2 to determine all of the drives. For example, And it with 1 to see if drive A: exists, with 2 for B:, 4 for C:, 8 for D:, etc. (See the example for a demonstration.)

Example:

' Tell the user which drives exist on the computer. Note how this example ' only checks up to drive D:, but it does establish the necessary pattern to use in general.
Dim driveflags As Long ' receives the flags identifying valid drives

' Get the valid logical drives on the computer.
driveflags = GetLogicalDrives()

' Test the returned value to see if drives A: through D: exist.
If (driveflags And 1) = 1 Then Debug.Print "Drive A: exists."
If (driveflags And 2) = 2 Then Debug.Print "Drive B: exists."
If (driveflags And 4) = 4 Then Debug.Print "Drive C: exists."
If (driveflags And 8) = 8 Then Debug.Print "Drive D: exists."
' And so on....

See Also: GetLogicalDriveStrings
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetLogicalDriveStrings Function

Declare Function GetLogicalDriveStrings Lib "kernel32.dll" Alias "GetLogicalDriveStringsA" (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long

Platforms: Win 95/98, Win NT

GetLogicalDriveStrings determines the valid logical drives on the computer and places the names of their root directories into the string passed as lpBuffer. Each root directory name in the buffer is separated by a null character, and the entire string ends in two null characters. For example, if only drives A: and C: exist, the string will be "a:\(null)c:\(null)(null)", where (null) represents the null character. The function returns 0 if an error occurred, or the length of the string placed in lpBuffer if successful.

nBufferLength
   The size of the buffer string passed as lpBuffer.
lpBuffer
   A string which receives the names of all the logical drives. This must have enough room to receive the string.

Example:

' List the names of all the root directories. Since each entry in the string takes
' four characters (three for the name and one for the null), we can "count by fours"
' through the string until we reach the end. This frees us from worrying about
' nulls.
Dim drivenames As String  ' receives list of root names
Dim thisdrive As String  ' buffer for one extracted root directory name
Dim c As Long  ' counter variable
Dim slength As Long  ' receives length of returned string

' Make enough room in the buffer to receive the drive names.
drivenames = Space(255)  ' more than enough room
' Get the root directory names of all logical drives.
slength = GetLogicalDriveStrings(255, drivenames)  ' drivenames now holds the list
' Count by fours to extract the names of each drive.
For c = 1 To slength Step 4  ' loop with an increment of 4
   thisdrive = Mid(drivenames, c, 3)  ' extract a 3-character string X:\ (X is the
   Debug.Print thisdrive  ' display the drive name
Next c

See Also: GetLogicalDrives
Category: Files
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getlogicaldrivestrings.html
GetMenu Function

Declare Function GetMenu Lib "user32.dll" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetMenu identifies what menu is assigned to a given window. This menu appears as the window's menu bar, although programatically the menu bar is just the top menu in the menu heirarchy.

Return Value

The function returns a handle to the menu assigned to the specified window. If the window has no menu, the function returns 0. If the window specified in the parameter list is a child window, the function's return value is meaningless.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to get the menu of.

Example

Before running this example, use the Menu Editor utility to create a small menu system on Form1. It doesn't matter what the menus look like, but some sort of menus are necessary.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetMenu Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_SUBMENU = &H4
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFS_CHECKED = &H8
Public Declare Function GetMenuItemInfo Lib "user32.dll" Alias "GetMenuItemInfoA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
    MENUITEMINFO) As Long

' When button Command1 is pressed, output the structure of the entire menu system
' of Form1 to the Debug window. The entire menu heirarchy is displayed, and any
' that are checked are identified. A recursive subroutine is used to output the
' contents
' of each individual menu, calling itself whenever a submenu is found.

' *** Place the following code inside a module. ***
' This function performs the recursive output of the menu structure.
Public Sub IterateThroughItems(ByVal hMenu As Long, ByVal level As Long)
    ' hMenu is a handle to the menu to output
    ' level is the level of recursion, used to indent submenu items
    Dim itemcount As Long    ' the number of items in the specified menu
    Dim c As Long            ' loop counter variable
    Dim mii As MENUITEMINFO  ' receives information about each item
    Dim retval As Long       ' return value
    ' Count the number of items in the menu passed to this subroutine.
    itemcount = GetMenuItemCount(hMenu)
    ' Loop through the items, getting information about each one.
    With mii
        .cbSize = Len(mii)
        .fMask = MIIM_STATE Or MIIM_TYPE Or MIIM_SUBMENU
    For c = 0 To itemcount - 1

' Make room in the string buffer.
.dwTypeData = Space(256)
.cch = 256
' Get information about the item.
retval = GetMenuItemInfo(hMenu, c, 1, mii)
' Output a line of information about this item.
If mii.fType = MFT_SEPARATOR Then
    ' This is a separator bar.
    Debug.Print "   " & String(3 * level, ".") & "-----"
Else
    ' This is a text item.
    ' If this is checked, display (X) in the margin.
    Debug.Print IIf(.fState And MFS_CHECKED, "(X)", "");
    ' Display the text of the item.
    Debug.Print String(3 * level, ".") &
    Left(.dwTypeData, .cch)
    ' If this item opens a submenu, display its contents.
    If .hSubMenu <> 0 Then
        IterateThroughItems .hSubMenu, level + 1
    End If
End If
Next c
End With
End Sub

' *** Place the following code inside Form1. ***
' When Command1 is clicked, output the entire contents of Form1's menu system.
Private Sub Command1_Click()
    Dim hMenu As Long ' handle to the menu bar of Form1
    ' Get a handle to Form1's menu bar.
    hMenu = GetMenu(Form1.hWnd)
    ' Use the above function to output its contents.
    IterateThroughItems hMenu, 0
End Sub

See Also

GetSystemMenu

Category

Menus

Back to the Function list.
Back to the Reference section.
GetMenuItemCount Function

Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetMenuItemCount counts the number of items in a menu. Keep in mind that the position indexes of menu items are zero based. This means that if there are \( n \) items in the menu, their position indexes go from 0 to \( n - 1 \).

Return Value

If successful, the function returns the number of items in the menu. If an error occurred, the function returns -1 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

\( hMenu \)

A handle to the menu to count the items of.

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
' There's quite a few declarations for this example, but it's worth it!
Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal
Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long

Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long

Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type

Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8

Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1

Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Public Const GWL_WNDPROC = -4

Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116
Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.

Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

*** Place the following code in a module. ***

Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public ontop As Boolean  ' identifies if Form1 is always on top or not

The following function acts as Form1's window procedure to process messages.
Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim hSysMenu As Long     ' handle to Form1's system menu
    Dim mii As MENUITEMINFO  ' menu item information for Always On Top
    Dim retval As Long       ' return value

    Select Case uMsg
        Case WM_INITMENU
            ' Before displaying the system menu, make sure that the Always On Top option is properly checked.
            hSysMenu = GetSystemMenu(hwnd, 0)
            With mii
                ' Size of the structure.
                .cbSize = Len(mii)
                ' Only use what needs to be changed.
                .fMask = MIIM_STATE
                ' If Form1 is now always on top, check the item.
                .fState = MFS_ENABLED Or IIf(ontop, MFS_CHECKED, 0)
            End With
            retval = SetMenuItemInfo(hSysMenu, 1, 0, mii)
            WindowProc = 0
        Case WM_SYSCOMMAND
            ' If Always On Top (ID = 1) was selected, change the on top/not on top setting of Form1 to match.
            If wParam = 1 Then
                ' Reverse the setting and make it the current one.
                ontop = Not ontop
                retval = SetWindowPos(hwnd, IIf(ontop, HWND_TOPMOST, HWND_NOTOPMOST), 0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
            End If
            WindowProc = 0
        Else
            ' Some other item was selected. Let the previous window procedure process it.
            WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
    End If

Windows API Guide: GetMenuItemCount Function

Case Else
  ' If this is some other message, let the previous procedure handle it.
  WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End Select
End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the new window procedure.
Private Sub Form_Load()
  Dim hSysMenu As Long     ' handle to the system menu
  Dim count As Long        ' the number of items initially on the menu
  Dim mii As MENUITEMINFO  ' describes a menu item to add
  Dim retval As Long       ' return value

  ' Get a handle to the system menu.
  hSysMenu = GetSystemMenu(Form1 hWnd, 0)
  ' See how many items are currently in it.
  count = GetMenuItemCount(hSysMenu)

  ' Add a separator bar and then Always On Top to the system menu.
  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .ftype = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With
  ' Add the separator to the end of the system menu.
  retval = InsertMenuItem(hSysMenu, count, 1, mii)

  ' Likewise, add the Always On Top command.
  With mii
    .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
    ' This is a regular text item.
    .fType = MFT_STRING
    ' The option is enabled.
    .fState = MFS_ENABLED
    ' It has an ID of 1 (this identifies it in the window procedure).
    .wID = 1
    ' The text to place in the menu item.
    .dwTypeData = "&Always On Top"
    .cch = Len(.dwTypeData)
  End With
  ' Add this to the bottom of the system menu.
  retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)
End Sub
' Set the custom window procedure to process Form1's messages.
ontop = False
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' Before unloading, restore the default system menu and remove the 
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value

    ' Replace the previous window procedure to prevent crashing.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)

    ' Remove the modifications made to the system menu.
    retval = GetSystemMenu(Form1.hWnd, 1)
End Sub

Category

Menus

Back to the Function list.
Back to the Reference section.

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getmenuitemcount.html
GetMenuItemInfo Function

Declare Function GetMenuItemInfo Lib "user32.dll" Alias "GetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetMenuItemInfo retrieves information about an item on a menu. The type of information retrieved is determined by the flags specified by the fMask data member of the structure passed as the lpmii parameter.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- **hMenu**
  A [handle](http://216.26.168.92/vbapi/ref/g/handleinfo.html) to the menu that contains the item to get information about.

- **uItem**
  Identifies the menu item to get information about. This could be either a position or a menu item identifier, depending on fByPosition.

- **fByPosition**
  If this is a non-zero value, uItem indicates the item by using its zero-based position. (For example, the first item in the menu has a position of 0.) If this is zero, then uItem is the unique menu item identifier of the item.

- **lpmii**
  Receives information about the menu item. Before calling GetMenuItemInfo, the cbSize and fMask data members of
this structure must be initialized to specify what information about the item to retrieve. If getting the text of the menu item, the `dwTypeData` and `cch` data members must also be initialized to receive the string. (See the example below for a demonstration of this.)

**Example**

Before running this example, use the Menu Editor utility to create a small menu system on Form1. It doesn't matter what the menus look like, but some sort of menus are necessary.

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getmenuiteminfo.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetMenu Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
  cbSize As Long
  fMask As Long
  fType As Long
  fState As Long
  wID As Long
  hSubMenu As Long
  hbmpChecked As Long
  hbmpUnchecked As Long
  dwItemData As Long
  dwTypeData As String
  cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_SUBMENU = &H4
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFS_CHECKED = &H8
Public Declare Function GetMenuItemInfo Lib "user32.dll" Alias "GetMenuItemInfoA" (ByVal _
  hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
  MENUITEMINFO) As Long

' When button Command1 is pressed, output the structure of the entire menu system
' of Form1 to the Debug window. The entire menu hierarchy is displayed, and any
' items
' that are checked are identified. A recursive subroutine is used to output the contents
' of each individual menu, calling itself whenever a submenu is found.

' *** Place the following code inside a module. ***
' This function performs the recursive output of the menu structure.
Public Sub IterateThroughItems(ByVal hMenu As Long, ByVal level As Long)
  ' hMenu is a handle to the menu to output
  ' level is the level of recursion, used to indent submenu items

Dim itemcount As Long    ' the number of items in the specified menu
Dim c As Long            ' loop counter variable
Dim mii As MENUITEMINFO  ' receives information about each item
Dim retval As Long       ' return value

' Count the number of items in the menu passed to this subroutine.
itemcount = GetMenuItemCount(hMenu)

' Loop through the items, getting information about each one.
With mii
    .cbSize = Len(mii)
    .fMask = MIIM_STATE Or MIIM_TYPE Or MIIM_SUBMENU
    For c = 0 To itemcount - 1
        ' Make room in the string buffer.
        .dwTypeData = Space(256)
        .cch = 256
        ' Get information about the item.
        retval = GetMenuItemInfo(hMenu, c, 1, mii)
        ' Output a line of information about this item.
        If mii.fType = MFT_SEPARATOR Then
            ' This is a separator bar.
            Debug.Print "   " & String(3 * level, ".") & "------"
        Else
            ' This is a text item.
            ' If this is checked, display (X) in the margin.
            Debug.Print IIf(.fState And MFS_CHECKED, "(X)", "") & "

            ' Display the text of the item.
            Debug.Print String(3 * level, ".") &
            Left(.dwTypeData, .cch)
        End If
        If .hSubMenu <> 0 Then
            IterateThroughItems .hSubMenu, level + 1
        End If
    Next c
End With
End Sub

*** Place the following code inside Form1. ***
' When Command1 is clicked, output the entire contents of Form1's menu system.
Private Sub Command1_Click()
    Dim hMenu As Long    ' handle to the menu bar of Form1

    ' Get a handle to Form1's menu bar.
    hMenu = GetMenu(Form1 hWnd)
    ' Use the above function to output its contents.
    IterateThroughItems hMenu, 0
End Sub

See Also
SetMenuItemInfo

Category

Menus

Back to the Function list.
Back to the Reference section.
GetNumberFormat Function

Declare Function GetNumberFormat Lib "kernel32.dll" Alias "GetNumberFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpValue As String, lpFormat As Any, ByVal lpNumberStr As String, ByVal cchNumber As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.5 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetNumberFormat formats a number for display. By default, the function formats the number according to the specified locale's settings. However, custom formatting preferences can instead be used. The end result of GetNumberFormat is a number displayed according to the user's preferences.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of characters copied into the string passed as lpNumberStr, not including the terminating null character.

Visual Basic-Specific Issues

When passing 0 for the lpFormat parameter, the expression ByVal CLng(0) must be used. See the example code for a demonstration of this.

Parameters

**Locale**

The locale identifier of the locale to format the number according to. This identifier could be generated by the MAKELCID macro. Alternatively, this could be one of the following flags specifying a locale:

- LOCALE_SYSTEM_DEFAULT
  - The system's default locale.
- LOCALE_USER_DEFAULT
  - The user's default locale.

**dwFlags**
If no structure is passed as \textit{lpFormat}, this parameter determines the settings used to format the number. If this is 0, the current locale settings are used. Or, this could be the following flag:

\textbf{LOCALE\_NOUSEROVERRIDE}

Use the system's default settings for the locale, regardless of any modifications the user may have made to it.

\textit{lpValue}

A string containing the number to format. The only allowable characters in this string are the digits 0-9 and at most a single decimal point character (\texttt{.}). If the number is negative, the first character in the string must be a minus sign character (\texttt{-}). Any other characters are invalid.

\textit{lpFormat}

To override the locale's formatting settings, pass a \textbf{NUMBERFMT} structure that contains the appropriate formatting information. To use the locale's settings instead, pass 0 for this parameter.

\textit{lpNumberStr}

String that receives the null-terminated formatted number string. This string must have enough room to receive the string.

\textit{cchNumber}

The number of characters in the string passed as \textit{lpNumberStr}.

\section*{Constant Definitions}

\begin{itemize}
\item \textbf{Const} LOCALE\_SYSTEM\_DEFAULT = \&H400
\item \textbf{Const} LOCALE\_USER\_DEFAULT = \&H800
\item \textbf{Const} LOCALE\_NOUSEROVERRIDE = \&H80000000
\end{itemize}

\section*{Example}

\begin{verbatim}
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type NUMBERFMT
    NumDigits As Long
    LeadingZero As Long
    Grouping As Long
    lpDecimalSep As String
    lpThousandSep As String
    NegativeOrder As Long
End Type
Public Declare Function GetNumberFormat Lib "kernel32.dll" Alias "GetNumberFormatA" _
    (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpValue As String, _
    lpFormat As Any, ByVal lpNumberStr As String, ByVal cchNumber As Long) As Long
Const LOCALE\_USER\_DEFAULT = \&H800

' Display the number -1,234,567.89 according to two formatting rules.
' 1. Use the format specified by the current user locale.
' 2. Use a custom format specified by a structure passed to the function.
Dim nft As NUMBERFMT ' custom formatting settings
Dim formatted As String ' receives the formatted number strings
Dim strlen As Long ' the length of the formatted string
\end{verbatim}
' Display the number formatted according to the current locale.
formatted = Space(256)
strlen = GetNumberFormat(LOCALE_USER_DEFAULT, 0, "-1234567.89", ByVal CLng(0), _
            formatted, Len(formatted))
formatted = Left(formatted, strlen)
Debug.Print "User locale format: "; formatted

' Now display according the format we specify below.
With nft
    ' Display three digits after the decimal point.
    .NumDigits = 3
    ' Display zeros after the decimal point.
    .LeadingZero = 1
    ' Group every three digits to the left of the decimal.
    .Grouping = 3
    ' Use a comma to as the decimal point (like they do in France and Spain).
    .lpDecimalSep = ","
    ' Likewise, use a period as the grouping separator.
    .lpThousandSep = "."
    ' Put the negative sign immediately after the number.
    .NegativeOrder = 3
End With
formatted = Space(256)
strlen = GetNumberFormat(LOCALE_USER_DEFAULT, 0, "-1234567.89", nft, formatted, Len(formatted))
formatted = Left(formatted, strlen)
Debug.Print "Custom format: "; formatted

See Also

GetCurrencyFormat

Category

National Language Support

Back to the Function list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getnumberformat.html
GetOpenFileName Function

Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" (lpofn As OPENFILENAME) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetOpenFileName opens the Open File common dialog dialog box. All of the settings for creating the dialog, as well as the data returned from it, are placed into the structure passed as \( lpofn \). Note that this function does not actually open any file; it merely prompts the user for a file or files to open, and those filenames are given to your program.

Return Value

If the user selects one or more files in the dialog box, the function returns a nonzero value. If an error occurred, or if the user merely clicked Cancel, the function returns zero. Use CommDlgExtendedError to get the error code.

Visual Basic-Specific Issues

None.

Parameters

\( lpofn \)

The parameters used to open the dialog box. Also receives the returned filename(s) and other information about what the user selected.

Example

Use the Open File common dialog box to prompt the user for a *.txt file. The user's selection is then displayed (the filename, not the file itself). The dialog box opens when the user clicks button Command1. So, to use this example, place a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getopenfilename.html).

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

```vbnet
Public Type OPENFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustomFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type
```

Public Const OFN_FILEMUSTEXIST = &H1000
Public Const OFN_HIDEREADONLY = &H4
Public Const OFN_PATHMUSTEXIST = &H800
Public Declare Function GetOpenFileName Lib "comdlg32.dll" Alias "GetOpenFileNameA" (lpofn As OPENFILENAME) As Long

*** Place the following code inside a form window.

```vbnet
Private Sub Command1_Click()
    Dim filebox As OPENFILENAME  ' open file dialog structure
    Dim fname As String          ' filename the user selected
    Dim result As Long           ' result of opening the dialog

    ' Configure how the dialog box will look
    With filebox
        ' Size of the structure.
        .lStructSize = Len(filebox)
        ' Handle to window opening the dialog.
        .hwndOwner = Me.hWnd
        ' Handle to calling instance (not needed).
        .hInstance = 0
        ' File filters to make available: Text Files and All Files
        .lpstrFilter = "Text Files (*.txt)" & vbNullChar & "*.txt" & vbCrLf & _
                       "All Files (*.*)" & vbNullChar & "*.*" & vbCrLf
    End With

    result = Call(GetOpenFileName, filebox)
    If result = 1 Then
        fname = filebox.lpstrFile
    End If
End Sub
```
'lpstrCustomFilter is ignored -- unused string
.nMaxCustomFilter = 0
'Default filter is the first one (Text Files, in this case).
.nFilterIndex = 1
'No default filename. Also make room for received
'path and filename of the user's selection.
.lpstrFile = Space(256) & vbNullChar
.nMaxFile = Len(.lpstrFile)
'Make room for filename of the user's selection.
.lpstrFileTitle = Space(256) & vbNullChar
.nMaxFileTitle = Len(.lpstrFileTitle)
'Initial directory is C:\.
.lpstrInitialDir = "C:\" & vbNullChar
'Title of file dialog.
.lpstrTitle = "Select a File" & vbNullChar
'The path and file must exist; hide the read-only box.
.flags = OFN_PATHMUSTEXIST Or OFN_FILEMUSTEXIST Or OFN_HIDEREADONLY
'The rest of the options aren't needed.
.nFileOffset = 0
.nFileExtension = 0
'.lpstrDefExt is ignored -- unused string
.lCustData = 0
.lpfnHook = 0
'.lpTemplateName is ignored -- unused string
End With

' Display the dialog box.
result = GetOpenFileName(filebox)
If result <> 0 Then
    ' Remove null space from the file name.
    fname = Left(filebox.lpstrFile, InStr(filebox.lpstrFile, vbNullChar) - 1)
    Debug.Print "The selected file: "; fname
End If
End Sub

See Also

GetSaveFileName

Category

Common Dialog

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: September 24, 2000
GetParent Function

Declare Function GetParent Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetParent returns the handle of the parent window of another window. For example, the parent of a button would normally be the form window it is in. If successful, the function returns a handle to the parent window. If it fails (for example, trying to find the parent of a non-window), it returns 0.

hwnd
The handle of the window to find the parent of.

Example:

' Figure out which frame, Frame1 or Frame2, the button Command1
' is located on -- it is considered the child of the frame it is in
Dim parenthwnd As Long  ' button's parent window
parenthwnd = GetParent(Command1.hWnd)  ' get the button's parent window
If parenthwnd = Frame1.hWnd Then Debug.Print "The button is inside Frame 1."
If parenthwnd = Frame2.hWnd Then Debug.Print "The button is inside Frame 2."

See Also: SetParent
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetPixel Function

Declare Function GetPixel Lib "gdi32.dll" (ByVal hdc As Long, ByVal nXPos As Long, ByVal nYPos As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetPixel determines the color of a specific pixel on a device. The function finds the RGB value of the pixel which is checked.

Return Value

If an error occurred, the function returns &HFFFF. If successful, the function returns the RGB value of the pixel which was checked.

Visual Basic-Specific Issues

None.

Parameters

- \textit{hdc}
  
  A handle to a device context to the device to check a pixel of.

- \textit{nXPos}
  
  The x-coordinate of the pixel to determine the color of.
**nYPos**

The y-coordinate of the pixel to determine the color of.

## Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getpixel.html).

' Fill all of window Form1 with the color which is located at coordinate (75,100). Fill the window by filling its rectangle using a brush derived from the color of the pixel.
Dim rgbVal As Long  ' RGB color of the pixel
Dim hBrush As Long  ' handle to the brush we'll create
Dim winrect As RECT  ' rectangle of Form1
Dim retval As Long  ' return value

' Determine the color of the pixel on Form1 at (75,100).
rgbVal = GetPixel(Form1.hDC, 75, 100)

' Get the dimensions of the rectangle of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)

' Create a solid brush of that color.
hBrush = CreateSolidBrush(rgbVal)

' Use that brush to fill in all of Form1 (its entire rectangle).
retval = FillRect(Form1.hDC, winrect, hBrush)

' Delete the brush we created to save resources.
retval = DeleteObject(hBrush)

## See Also

[SetPixel](http://216.26.168.92/vbapi/ref/g/setpixel.html), [SetPixelV](http://216.26.168.92/vbapi/ref/g/setpixelv.html)

## Category

Bitmaps

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** September 2, 1999
GetPolyFillMode Function

Declare Function GetPolyFillMode Lib "gdi32.dll" (ByVal hdc As Long)
As Long

Platforms: Win 32s, Win 95/98, Win NT

GetPolyFillMode determines how a given device fills polygonal areas and shapes. These two modes only differ in how they handle complex overlapping polygons (i.e., polygons whose boundaries criss-cross themselves). The function returns 0 if an error occurred, or exactly one of the following flags if successful:

ALTERNATE = 1
The device alternates between filling and not filling contiguous sections whose boundaries are determined by the edge(s) of the polygon crossing through the polygon's interior.

WINDING = 2
Any section inside the polygon is filled, regardless of any intra-polygonal boundaries and edges.

hdc
A device context to the device to find the polygon filling mode of.

Example:

' Display the current polygon fill mode of window Form1.
Dim fillmode As Long  ' receives fill mode
fillmode = GetPolyFillMode(Form1.hDC)  ' get the polygon fill mode
If fillmode = ALTERNATE Then
    Debug.Print "Form1 currently uses alternating filling."
Else
    Debug.Print "Form1 currently uses winding filling."
End If

See Also: SetPolyFillMode
Category: Regions
GetPrivateProfileInt Function

Declare Function GetPrivateProfileInt Lib "kernel32.dll" Alias "GetPrivateProfileIntA" (ByVal lpApplicationName As String, ByVal lpKeyName As String, ByVal nDefault As Long, ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetPrivateProfileInt reads an integer value from any INI file. The parameters passed to the function specify which value will be read from. If successful, the function returns the value read. If the value you specify does not exist or is a string (i.e., not a number), the value specified as nDefault is returned. Note that INI file support is only provided in Windows for backwards compatibility; using the registry to store information is preferred.

lpApplicationName
The header of the INI file section the value is in.

lpKeyName
The name of the value to read.

nDefault
The value to return if a valid value cannot be read. Make it something that would definitely not be read, such as -1.

lpFileName
The filename of the INI file to read from.

Example:

' Read the "version" value under the "[programinfo]" section
' of the INI file C:\MyProgram\config.ini
Dim version As Long ' receives the value returned from the INI file

' Read the value from the INI file, returning -1 if it can't find the value
version = GetPrivateProfileInt("programinfo", "version", -1, "C:\MyProgram\config.ini")
' Display the result
If version = -1 Then ' failure
   Debug.Print "Could not read the information from the INI file."
Else
   Debug.Print "Version number:"; version
End If
See Also: GetPrivateProfileString, GetProfileInt
Category: INI Files
GetPrivateProfileString Function

Declare Function GetPrivateProfileString Lib "kernel32.dll" Alias "GetPrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpDefault As String, ByVal lpReturnedString As String, ByVal nSize As Long, ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetPrivateProfileString reads an string value from an INI file. The parameters passed to the function specify which value will be read from. The function always returns the length in characters of the string put into the variable passed as lpReturnedString. If the function was successful, the string read from the INI file will be put into lpReturnedString. If not, it will instead receive the string given as lpDefault. Note that INI file support is only provided in Windows for backwards compatibility; using the registry to store information is preferred.

lpApplicationName
  The header of the INI file section the value is in.
lpKeyName
  The name of the value to read.
lpDefault
  The value to return if a valid value cannot be read. Make it something that would definitely not be read, such as "(error)".
lpReturnedString
  A fixed-length string that will receive either the string read from the file or lpDefault.
nSize
  The length in characters of lpReturnedString.
lpFileName
  The filename of the INI file to read from.

Example:

' Read the "username" value under the [default] section of
' the INI file C:\MyProgram\config.ini. The default value is "anonymous".
Dim uname As String  ' receives the value read from the INI file
Dim slength As Long  ' receives length of the returned string
uname = Space(255)  ' provide enough room for the function to put the value into the buffer
' Read from the INI file
slength = GetPrivateProfileString("default", "username", "anonymous", uname, 255, "C:\MyProgram\config.ini")
uname = Left(uname, slength)  ' extract the returned string from the buffer
Debug.Print "User's name: "; uname
See Also: GetPrivateProfileInt, GetProfileString, WritePrivateProfileString
Category: INI Files
GetProfileInt Function

Declare Function GetProfileInt Lib "kernel32.dll" Alias "GetProfileIntA" (ByVal lpAppName As String, ByVal lpKeyName As String, ByVal nDefault As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Note: This function is primarily for backwards compatibility with 16-bit programs. 32-bit Windows programs should typically store data in the Registry instead.

Description & Usage

GetProfileInt reads an integer value from the WIN.INI file. The parameters passed to the function specify which value will be read from. This is basically a watered-down version of GetPrivateProfileInt because, unlike it, this function only works with WIN.INI.

Return Value

If successful, the function returns the integer value read from the INI file. If the specified key could not be found, the function returns the value passed as nDefault.

Visual Basic-Specific Issues

None.

Parameters

lpAppName

The header of the WIN.INI file section the value is in.

lpKeyName
The name of the value to read.

\textit{nDefault}

The value to return if a valid value cannot be read. This should typically be a default setting for the value, or something that would indicate that the value could not be read.

**Example**

Read the value "WallpaperStyle" from the [Desktop] section of the WIN.INI file and display it. To use this example, place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetProfileInt Lib "kernel32.dll" Alias "GetProfileIntA" (ByVal lpAppName As String, ByVal lpKeyName As String, ByVal nDefault As Long) As Long

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim value As Long  ' value read from WIN.INI
    ' Read the desired value.
    value = GetProfileInt("Desktop", "WallpaperStyle", -1)
    ' Display the result.
    If value = -1 Then
        Debug.Print "Could not read the value!"
    Else
        Debug.Print "WallpaperStyle ="; value
    End If
End Sub

**See Also**

GetPrivateProfileInt, GetProfileString

**Category**

INI Files

Go back to the Function listing.
Go back to the Reference section index.
GetRgnBox Function

Declare Function GetRgnBox Lib "gdi32.dll" (ByVal hRgn As Long, lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetRgnBox determines the bounding rectangle of a given region. The bounding rectangle is the smallest possible rectangle which fully encompasses the region. The bounding rectangle is placed in the structure passed as lpRect. The function returns 0 if an error occurred, or exactly one of the following flags specifying the shape of the input region:

COMPLEXREGION = 3
    The region is not empty but is not rectangular.
NULLREGION = 1
    The region is empty, i.e., null.
SIMPLEREGION = 2
    The region is rectangular in shape.

hRgn
    A handle to the region to get the bounding rectangle of.
lpRect
    Receives the coordinates of the region's bounding rectangle.

Example:

' Fill a triangular region on Form1 in dark gray and fill its bounding rectangle (behind it) in light gray.
Dim triangle(0 To 2) As POINT_TYPE  ' vertices of triangular region
Dim hRgn As Long  ' handle to the triangular region
Dim hLightBrush As Long, hDarkBrush As Long  ' handles to the two brushes
Dim bounding As RECT  ' receives bounding rectangle of region
Dim retval As Long  ' generic return value

' Create the triangular region.
triangle(0).x = 150: triangle(0).y = 100  ' point #1: (150,100)
triangle(1).x = 200: triangle(1).y = 150  ' point #2: (200,150)
triangle(2).x = 175: triangle(2).y = 200  ' point #3: (175,200)
hRgn = CreatePolygonRgn(triangle(0), 3, ALTERNATE)  ' create the region
'Get handles to the light and dark gray solid stock brushes to use.
hLightBrush = GetStockObject(LTGRAY_BRUSH)
hDarkBrush = GetStockObject(DKGRAY_BRUSH)

'Calculate the triangular region's bounding rectangle.
retval = GetRgnBox(hRgn, bounding)  ' now bounding is the bounding rectangle
'Fill the bounding rectangle in light gray, the region in dark gray.
retval = FillRect(Form1.hDC, bounding, hLightBrush)
retval = FillRgn(Form1.hDC, hRgn, hDarkBrush)

'Delete the region to free up resources.
retval = DeleteObject(hRgn)

Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetProfileString Function

Declare Function GetProfileString Lib "kernel32.dll" Alias "GetProfileStringA"
(ByVal lpAppName As String, ByVal lpKeyName As String, ByVal lpDefault As String,
ByVal lpReturnedString As String, ByVal nSize As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetProfileString reads an string value from the WIN.INI file. The parameters passed to the function specify which value will be read from. The function always returns the length in characters of the string put into the variable passed as lpReturnedString. If the function was successful, the string read from the INI file will be put into lpReturnedString. If not, it will instead receive the string given as lpDefault. This function is basically a watered-down version of GetPrivateProfileString because it, unlike this function, works with all INI files. Note that INI file support is only provided in Windows for backwards compatibility; using the registry to store information is preferred.

lpAppName
   The header of the INI file section the value is in.
lpKeyName
   The name of the value to read.
lpDefault
   The value to return if a valid value cannot be read. Make it something that would definitely not be read, such as "(error)".
lpReturnedString
   A fixed-length string that will receive either the string read from the file or lpDefault.
nSize
   The length in characters of lpReturnedString.

Example:

' Read the value "Wallpaper" from under the [Desktop] section
' of WIN.INI. If an error occurs, the function will return "(error)"
Dim wallpaper As String  ' receives string read from WIN.INI
Dim slength As Long  ' receives length of string read from WIN.INI

wallpaper = Space(255)  ' make room in the buffer to receive the string read from WIN.INI
' Read the string from WIN.INI
slength = GetProfileString("Desktop", "Wallpaper", "(error)", buffer, 255)
wallpaper = Left(wallpaper, slength)  ' extract the returned string from the buffer
If wallpaper = "(error)" Then
   Debug.Print "Could not read information from WIN.INI."
Else
   Debug.Print "Wallpaper file: " ; wallpaper
End If

See Also: GetPrivateProfileString, GetProfileInt, WriteProfileString

Category: INI Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Go back to the Windows API Guide home page.
E-mail: rogue953@st-louis.crosswinds.net
This page is at http://www.vbapi.com/ref/g/getprofilestring.html
GetProp Function

Declare Function GetProp Lib "user32.dll" Alias "GetPropA" (ByVal hWnd As Long, ByVal lpString As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetProp retrieves the value of one of the window properties of a window. The function obtains a handle to whatever object was associated with the window property.

Return Value

If successful, the function returns a handle to the object associated with the window property. If the desired window property could not be found, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to the window to read a window property of.

lpString
The name of the window property to read.

Example

' This code is licensed according to the terms and conditions listed here.
' Read the "LookupFile" property of window Form1 and display it.
' This property is set by the example for the SetProp function --
' the value is a string specifying a filename.
Dim hData As Long, pData As Long  ' handle and pointer to the property's data
Dim strvalue As String  ' receives the value of the window property (in this case)
Dim retval As Long  ' return value

' Get a handle to the window property's data.
hData = GetProp(Form1.hWnd, "LookupFile")
' If that property didn't exist, alert the user.
If hData = 0 Then
  Debug.Print "There is no property called 'LookupFile'."
Else
  ' Get a pointer to the data retrieved.
pData = GlobalLock(hData)
  ' Copy the string data into the string.
  strvalue = Space(lstrlen(pData))
  retval = lstrcpy(strvalue, pData)
  ' Unlock the data block.
  retval = GlobalUnlock(hData)
  ' Display the string gotten from the data block.
  Debug.Print "LookupFile = "; strvalue
End If

See Also

SetProp

Category

Window Properties

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: December 24, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getprop.html
GetSaveFileName Function

Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA" (lpofn As OPENFILENAME) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

`GetSaveFileName` opens the Save File common dialog dialog box. All of the settings for creating the dialog, as well as the data returned from it, are placed into the structure passed as `lpofn`. Note that this function does not actually save any file; it merely prompts the user for a file to save, and that filename is given to your program.

Return Value

If the user selects a file in the dialog box, the function returns a nonzero value. If an error occurred, or if the user merely clicked Cancel, the function returns zero. Use `CommDlgExtendedError` to get the error code.

Visual Basic-Specific Issues

None.

Parameters

`lpofn`

The parameters used to open the dialog box. Also receives the returned filename and other information about what the user selected.

Example

Use the Save File common dialog box to prompt the user for a *.txt file. The user's selection is then displayed (the filename, not the file itself). The dialog box opens when the user clicks button Command1. So, to use this example, place a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Type OPENFILENAME
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustomFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type

Public Const OFN_HIDEREADONLY = &H4
Public Const OFN_OVERWRITEPROMPT = &H2
Public Const OFN_PATHMUSTEXIST = &H800

Public Declare Function GetSaveFileName Lib "comdlg32.dll" Alias "GetSaveFileNameA"
    (lpofn As OPENFILENAME) As Long

*** Place the following code inside a form window.

Private Sub Command1_Click()
    Dim filebox As OPENFILENAME ' save file dialog structure
    Dim fname As String ' filename the user selected
    Dim result As Long ' result of opening the dialog

    ' Configure how the dialog box will look
    With filebox
        ' Size of the structure.
        .lStructSize = Len(filebox)
        ' Handle to window opening the dialog.
        .hwndOwner = Me.hWnd
        ' Handle to calling instance (not needed).
        .hInstance = 0
        ' File filters to make available: Text Files and All Files
        .lpstrFilter = "Text Files (*.txt)" & vbCrLf & "All Files (*.*)" & vbCrLf
    End With

    result = GetSaveFileName lpofn As OPENFILENAME
    If result = 1 Then
        fname = filebox.lpstrFile
    End If
End Sub
'.lpstrCustomFilter is ignored -- unused string
.nMaxCustomFilter = 0
' Default filter is the first one (Text Files, in this case).
.nFilterIndex = 1
' No default filename. Also make room for received
' path and filename of the user's selection.
.lpstrFile = Space(256) & vbNullChar
.nMaxFile = Len(.lpstrFile)
' Make room for filename of the user's selection.
.lpstrFileTitle = Space(256) & vbNullChar
.nMaxFileTitle = Len(.lpstrFileTitle)
' Initial directory is C:\.
.lpstrInitialDir = "C:\" & vbNullChar
' Title of file dialog.
.lpstrTitle = "Select a File" & vbNullChar
' The path must exist. Hide the read-only box.
.flags = OFN_PATHMUSTEXIST Or OFN_HIDEREADONLY Or OFN_OVERWRITEPROMPT
' The rest of the options aren't needed.
.nFileOffset = 0
.nFileExtension = 0
'.lpstrDefExt is ignored -- unused string
.lCustData = 0
.lpfnHook = 0
'.lpTemplateName is ignored -- unused string
End With

' Display the dialog box.
result = GetSaveFileName(filebox)
If result <> 0 Then
  ' Remove null space from the file name.
  fname = Left(filebox.lpstrFile, InStr(filebox.lpstrFile, vbNullChar) - 1)
  Debug.Print "The selected file: "; fname
End If
End Sub

See Also

GetOpenFileName

Category

Common Dialog

Go back to the Function listing.
Go back to the Reference section index.
GetShortPathName Function

Declare Function GetShortPathName Lib "kernel32.dll" Alias "GetShortPathNameA" (ByVal lpszLongPath As String, ByVal lpszShortPath As String, ByVal cchBuffer As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Not Supported.

Description & Usage

GetShortPathName converts a long filename into the old-style 8.3 filenames. Although Windows allows the use of long filenames, DOS programs and 16-bit Windows programs must use the 8.3 equivalent. For example, the equivalent of ReallyLongFile.txt could be REALLY~1.TXT.

Return Value

If successful, the function returns the length of the string copied into lpszShortPath, not including the terminating null character. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

lpszLongPath
The complete long path and filename to convert.

lpszShortPath
Receives the 8.3 form of the filename, terminated by a null character. This string must already be sufficiently large to receive the 8.3 filename.

cchBuffer
The length of the string passed as lpszShortPath.
Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetShortPathName Lib "kernel32.dll" Alias "GetShortPathNameA"
_  (ByVal lpszLongPath As String, ByVal lpszShortPath As String, ByVal cchBuffer
_    As Long) As Long

' Find the short filename equivalent of "C:\My Documents\ReadMeFirst.txt"
Private Sub Command1_Click()
  Dim shortname As String  ' receives short-filename equivalent
  Dim slength As Long  ' receives length of short-filename equivalent

  ' Make room in the buffer to receive the 8.3 form of the filename.
  shortname = Space(256)
  ' Get the 8.3 form of the filename specified.
  slength = GetShortPathName("C:\My Documents\ReadMeFirst.txt", shortname, 256)
  ' Remove the trailing null and display the result.
  shortname = Left(shortname, slength)
  Debug.Print "Equivalent: "; shortname
End Sub

See Also

GetFullPathName

Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getshortpathname.html
GetStockObject Function

Declare Function GetStockObject Lib "gdi32.dll" (ByVal nIndex As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetStockObject accesses one of Windows's stock pens, brushes, fonts, or palettes. This function provides fast access to these commonly used objects, instead of having to use more complicated functions. The function returns a handle to the pen, brush, font, or palette which the function accesses. Although the program isn't required to delete the handle using DeleteObject, doing so doesn't have any adverse effects.

nIndex

Exactly one of the following flags specifying which of the stock objects to create a handle to:

- ANSI_FIXED_FONT = 11
  The system's normal monospaced font.
- ANSI_VAR_FONT = 12
  The system's normal proportional-width font.
- BLACK_BRUSH = 4
  A solid black brush.
- BLACK_PEN = 7
  A solid black pen.
- DEFAULT_GUI_FONT = 17
  Win 95/98 only: The default font for user objects under Windows.
- DEFAULT_PALETTE = 15
  The default system palette.
- DEVICE_DEFAULT_FONT = 14
  Win NT only: A device-dependent font.
- DKGRAY_BRUSH = 3
  A solid dark gray brush.
- GRAY_BRUSH = 2
  A solid gray brush.
- HOLLOW_BRUSH = 5
  Same as NULL_BRUSH.
- LTGRAY_BRUSH = 1
  A solid light gray brush.
- NULL_BRUSH = 5
  A null brush; i.e., a brush that does not draw anything on the device.
- NULL_PEN = 8
  A null pen; i.e., a pen that does not draw anything on the device.
- OEM_FIXED_FONT = 10
  The Original Equipment Manufacturer's default monospaced font.
- SYSTEM_FIXED_FONT = 16
  The system monospaced font under pre-3.x versions of Windows.
- SYSTEM_FONT = 13
  The system font (used for most system objects under Windows).
WHITE_BRUSH = 0
A solid white brush.
WHITE_PEN = 6
A solid white pen.

Example:

' Draw a rectangle with a black border and light gray filled interior
' on window Form1. Use stock pens and brushes to do this.
Dim hbrush As Long, holdbrush As Long ' handles to stock brush & default brush
Dim hpen As Long, holdpen As Long ' handles to stock pen & default pen
Dim retval As Long ' return value

' Load the stock pen and brush needed for this operation
hpen = GetStockObject (BLACK_PEN) ' load the black solid pen
hbrush = GetStockObject (LTGRAY_BRUSH) ' load the light gray solid brush
' Select the two objects in Form1 and save the defaults
holdpen = SelectObject (Form1.hDC, hpen) ' select the pen
holdbrush = SelectObject (Form1.hDC, hbrush) ' select the brush
' Draw the rectangle using the pen and brush. The rectangle has corners (20,25)-(200,175)
retval = Rectangle (Form1.hDC, 20, 25, 200, 175)
' Restore Form1's previously selected pen and brush
retval = SelectObject (Form1.hDC, holdpen) ' reselect old pen
retval = SelectObject (Form1.hDC, holdbrush) ' reselect old brush
' Note that it is not necessary to delete hpen and hbrush, but we could if we wanted.

Category: Devices

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetSysColor Function

Declare Function GetSysColor Lib "user32.dll" (ByVal nIndex As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetSysColor returns the RGB value of one of the system colors. The system colors are the ones Windows uses to draw the typical widgets in the graphical user interface. For example, this includes the colors of title bars, scroll bars, the desktop, window borders, etc.

Return Value

If successful, the function returns the RGB value of the requested system color. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

nIndex

One of the following flags specifying which system color to get:

- COLOR_3DDKSHADOW
  The dark shadow color for 3D objects.
- COLOR_3DFACE, COLOR_BTNFACE
  The face color for 3D objects.
- COLOR_3DHILIGHT, COLOR_3DHIGHLIGHT, COLORBTNHLIGHT, COLOR_BTNHIGHLIGHT
  The highlight (opposite of shadow) color for 3D objects.
- COLOR_3DLIGHT
  The light (opposite of shadow) color for 3D objects.
- COLOR_3DSHADOW, COLOR_BTNSHADOW
The shadow color for 3D objects.

**COLOR_ACTIVEBORDER**
The active window border color.

**COLOR_ACTIVECAPTION**
The active window title bar color. **Windows 98, 2000:** The color of the left side of the active window title bar gradient, if the gradient effect is used.

**COLOR_APPWORKSPACE**
The background color of multiple document interface (MDI) windows.

**COLOR_BACKGROUND, COLOR_DESKTOP**
The desktop color.

**COLOR_BTNTEXT**
The text color for pushbuttons.

**COLOR_CAPTIONTEXT**
The color of window caption text, size boxes, and scroll bar arrow boxes.

**COLOR_GRADIENTACTIVECAPTION**
**Windows 98, 2000:** The color of the right side of the active window title bar gradient, if the gradient effect is used.

**COLOR_GRADIENTINACTIVECAPTION**
**Windows 98, 2000:** The color of the right side of an inactive window's title bar gradient, if the gradient effect is used.

**COLOR_GRAYTEXT**
The color for disabled (grayed-out) text.

**COLOR_HIGHLIGHT**
The color used to highlight selected items.

**COLOR_HIGHLIGHTTEXT**
The color used for the text of highlighted items.

**COLOR_HOTLIGHT**
**Windows 98, 2000:** The color of a hot-tracked item, which is executed with a single click.

**COLOR_INACTIVEBORDER**
The color of an inactive window's border.

**COLOR_INACTIVECAPTION**
The color of an inactive window's caption. **Windows 98, 2000:** The color of the right side of an inactive window's title bar gradient, if the gradient effect is used.

**COLOR_INACTIVECAPTIONTEXT**
The color of an inactive window's caption text.

**COLOR_INFOBK**
The background color for tooltip controls.

**COLOR_INFOTEXT**
The text color for tooltip controls.

**COLOR_MENU**
The background color of menus.

**COLOR_MENUTEXT**
The color of menu text.

**COLOR_SCROLLBAR**
The color of a scroll bar's gray area.

**COLOR_WINDOW**
The background color of a window.

**COLOR_WINDOWFRAME**
The color of a window frame.

**COLOR_WINDOWTEXT**
The color of text in a window.
Constant Definitions

Const COLOR_3DDKSHADOW = 21
Const COLOR_3DFACE = COLOR_BTNFACE
Const COLOR_3DHIGHLIGHT = COLOR_BTNHIGHLIGHT
Const COLOR_3DHILIGHT = COLOR_BTNHIGHLIGHT
Const COLOR_3DLIGHT = 22
Const COLOR_3DSHADOW = COLOR_BTNSHADOW
Const COLOR_ACTIVEBORDER = 10
Const COLOR_ACTIVECAPTION = 2
Const COLOR_APPWORKSPACE = 12
Const COLOR_BACKGROUND = 1
Const COLOR_BTNFACE = 15
Const COLOR_BTNHIGHLIGHT = 20
Const COLOR_BTNHILIGHT = COLOR_BTNHIGHLIGHT
Const COLOR_BTNSHADOW = 16
Const COLOR_BTNTEXT = 18
Const COLOR_CAPTIONTEXT = 9
Const COLOR_DESKTOP = COLOR_BACKGROUND
Const COLOR_GRADIENTACTIVECAPTION = 27
Const COLOR_GRADIENTINACTIVECAPTION = 28
Const COLOR_GRAYTEXT = 17
Const COLOR_HIGHLIGHT = 13
Const COLOR_HIGHLIGHTTEXT = 14
Const COLOR_HOTLIGHT = 26
Const COLOR_INACTIVEBORDER = 11
Const COLOR_INACTIVECAPTION = 3
Const COLOR_INACTIVECAPTIONTEXT = 19
Const COLOR_INFOBK = 24
Const COLOR_INFOTEXT = 23
Const COLOR_MENU = 4
Const COLOR_MENUTEXT = 7
Const COLOR_SCROLLBAR = 0
Const COLOR_WINDOW = 5
Const COLOR_WINDOWFRAME = 6
Const COLOR_WINDOWTEXT = 8

Example

Reverse the gradient colors in windows' title bars. In other words, the left-side gradient color is swapped with the right-side gradient color, for both active and inactive windows. Of course, this example won't work properly on Windows 95 or Windows NT 3.1 through 4.0, but you can still see how these two functions are used by looking at the source code.

This example runs when the user clicks command button Command1. So, to use this example, you must first place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Windows API Guide: GetSysColor Function

Public Declare Function GetSysColor Lib "user32.dll" (ByVal nIndex As Long) As Long
Public Declare Function SetSysColors Lib "user32.dll" (ByVal cElements As Long, _
    lpaElements As Long, lpaRgbValues As Long) As Long
Public Const COLOR_ACTIVECAPTION = 2
Public Const COLOR_GRADIENTACTIVECAPTION = 27
Public Const COLOR_GRADIENTINACTIVECAPTION = 28
Public Const COLOR_INACTIVECAPTION = 3

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
  Dim activeLeftColor As Long     ' color of left-side gradient of active window title bar
  Dim activeRightColor As Long    ' color of right-side gradient of active window title bar
  Dim inactiveLeftColor As Long   ' color of the left-side gradient of inactive window title bar
  Dim inactiveRightColor As Long  ' color of the right-side gradient of inactive window title bar
  Dim colorNames(0 To 3) As Long  ' identifiers of the system colors to change
  Dim colorRGBs(0 To 3) As Long   ' RGB values of the system colors to change
  Dim retval As Long              ' generic return value

  ' Get the RGB values of the colors used in title bars.
  activeLeftColor = GetSysColor(COLOR_ACTIVECAPTION)
  activeRightColor = GetSysColor(COLOR_GRADIENTACTIVECAPTION)
  inactiveLeftColor = GetSysColor(COLOR_INACTIVECAPTION)
  inactiveRightColor = GetSysColor(COLOR_GRADIENTINACTIVECAPTION)

  ' Load the arrays with the new values to assign. Note how we're switching the values used on the left and right sides of the two gradients.
  colorNames(0) = COLOR_ACTIVECAPTION
  colorRGBs(0) = activeRightColor
  colorNames(1) = COLOR_GRADIENTACTIVECAPTION
  colorRGBs(1) = activeLeftColor
  colorNames(2) = COLOR_INACTIVECAPTION
  colorRGBs(2) = inactiveRightColor
  colorNames(3) = COLOR_GRADIENTINACTIVECAPTION
  colorRGBs(3) = inactiveLeftColor

  ' Change the system color settings as specified above.
  retval = SetSysColors(4, colorNames(0), colorRGBs(0))
End Sub

See Also

SetSysColors

GetSystemDirectory Function

Declare Function GetSystemDirectory Lib "kernel32.dll" Alias "GetSystemDirectoryA"
(ByVal lpBuffer As String, ByVal nSize As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetSystemDirectory reads the path of Windows's System directory. This is where many important files for Windows are stored, including the API DLLs. Never assume this is "C:\Windows\System" because the Windows directory doesn't have to be called Windows. The path of the system directory is put into the string variable passed as lpBuffer. The function returns the length of the string returned if it is successful, or 0 if it failed.

lpBuffer
A string which receives the path of the system directory.

nSize
The length in characters of lpBuffer.

Example:

' Display the path of the system directory
Dim sysdir As String  ' receives path of system directory
Dim slength As Long  ' receives length of returned string
sysdir = Space(255)  ' make room in the buffer to receive the string
slength = GetSystemDirectory(sysdir, 255)  ' determine the system directory's path
sysdir = Left(sysdir, slength)  ' extract the returned string from the buffer
Debug.Print "System directory path: "; sysdir

See Also: GetWindowsDirectory
Category: System Information

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetSystemMenu Function

Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetSystemMenu obtains a handle to a window's system menu (also known as its control menu). This menu is what appears when the application icon on the title bar is clicked, or when the title bar itself is right-clicked. Once you get a handle to a window's system menu, you can use the other menu functions to add or remove items from it. However, note that all items on a system menu send the WM_SYSCOMMAND message to the parent window instead of the usual WM_COMMAND message. Alternately, GetSystemMenu could also be used to restore the system menu to its default appearance, removing any modifications done to it.

Return Value

If the bRevert parameter is zero, GetSystemMenu returns a handle to the window's system menu. If bRevert is nonzero, this function returns 0.

Visual Basic-Specific Issues

None.

Parameters

- **hWnd**
  A handle to the window to locate the system menu of.

- **bRevert**
  If zero, the function obtains a handle to the window's system menu. If nonzero, the function instead reverts the system menu back to its default appearance.
Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
' There's quite a few declarations for this example, but it's worth it!
Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
cbSize As Long
fMask As Long
fType As Long
fState As Long
wID As Long
hSubMenu As Long
hbmpChecked As Long
hbmpUnchecked As Long
dwItemData As Long
dwTypeData As String
cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1
Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewValue As Long) As Long
As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = -4
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal _
    lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam _
    As Long, ByVal lParam As Long) As Long
Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116

' Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.
' Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

' *** Place the following code in a module. ***

Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public ontop As Boolean  ' identifies if Form1 is always on top or not

' The following function acts as Form1's window procedure to process messages.
Public Function WindowProc ((ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam _
    As Long, ByVal lParam As Long) As Long
Dim hSysMenu As Long     ' handle to Form1's system menu
Dim mii As MENUITEMINFO  ' menu item information for Always On Top
Dim retval As Long       ' return value

Select Case uMsg
Case WM_INITMENU
    ' Before displaying the system menu, make sure that the Always On Top option is properly checked.
hSysMenu = GetSystemMenu(hwnd, 0)
With mii
    ' Size of the structure.
    .cbSize = Len(mii)
    ' Only use what needs to be changed.
    .fMask = MIIM_STATE
    ' If Form1 is now always on top, check the item.
    .fState = MFS_ENABLED Or IIf(ontop, MFS_CHECKED, 0)
End With
retval = SetMenuItemInfo(hSysMenu, 1, 0, mii)
WindowProc = 0
Case WM_SYSCOMMAND
    ' If Always On Top (ID = 1) was selected, change the on top/not on top setting of Form1 to match.
    If wParam = 1 Then
        ' Reverse the setting and make it the current one.
        ontop = Not ontop
        retval = SetWindowPos(hwnd, IIf(ontop, HWND_TOPMOST, HWND_NOTOPMOST), _
        0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
    End If
End Select

WindowProc = 0
Else
  ' Some other item was selected. Let the previous window
  ' procedure process it.
  WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End If
Case Else
  ' If this is some other message, let the previous procedure handle it.
  WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End Select
End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the
' new window procedure.
Private Sub Form_Load()
  Dim hSysMenu As Long     ' handle to the system menu
  Dim count As Long        ' the number of items initially on the menu
  Dim mii As MENUITEMINFO  ' describes a menu item to add
  Dim retval As Long       ' return value

  ' Get a handle to the system menu.
  hSysMenu = GetSystemMenu(Form1.hWnd, 0)
  ' See how many items are currently in it.
  count = GetMenuItemCount(hSysMenu)

  ' Add a separator bar and then Always On Top to the system menu.
  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With
  ' Add the separator to the end of the system menu.
  retval = InsertMenuItem(hSysMenu, count, 1, mii)

  ' Likewise, add the Always On Top command.
  With mii
    .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
    ' This is a regular text item.
    .fType = MFT_STRING
    ' The option is enabled.
    .fState = MFS_ENABLED
    ' It has an ID of 1 (this identifies it in the window procedure).
  End With
  retval = InsertMenuItem(hSysMenu, count, 1, mii)
End Sub
.wID = 1
   ' The text to place in the menu item.
   .dwTypeData = "&Always On Top"
   .cch = Len(.dwTypeData)

End With
   ' Add this to the bottom of the system menu.
   retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)

   ' Set the custom window procedure to process Form1's messages.
   ontop = False
   pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)

End Sub

' Before unloading, restore the default system menu and remove the
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
   Dim retval As Long
      ' return value

   ' Replace the previous window procedure to prevent crashing.
   retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)

   ' Remove the modifications made to the system menu.
   retval = GetSystemMenu(Form1.hWnd, 1)

End Sub

See Also

GetMenu

Category

Menus

Back to the Function list.
Back to the Reference section.

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g getsystemmenu.html
GetSystemMetrics Function

Declare Function GetSystemMetrics Lib "user32.dll" (ByVal nIndex As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetSystemMetrics returns information about various things in Windows. Most of these deal with the sizes of various objects, such as the screen, icons, cursors, etc. This function provides information about the system. The return value of the function depends on the value specified for nIndex. Keep in mind that all sizes (such as widths and heights) are measured in pixels. Also note that some of the metrics have slightly different meanings between Win NT and Win 95/98.

nIndex

Exactly one of the following flags specifying which piece of information to return:

SM_ARRANGE = 56

Win 95/98 only: Return the method used to display minimized windows. The return value is a combination of two of the following flags, one specifying a starting position for the minimized icons and another specifying the direction in which new ones are added:

ARW_BOTTOMLEFT = 0
Start placing the icons in the bottom-left corner of the screen.

ARW_BOTTOMRIGHT = 1
Start placing the icons in the bottom-right corner of the screen.

ARW_DOWN = 4
Add new icons below existing ones.

ARW_HIDE = 8
Do not place the icons anywhere on the screen (i.e., hide them).

ARW_LEFT = 0
Add new icons to the left of existing ones.

ARW_RIGHT = 4
Add new icons to the right of existing ones.

ARW_STARTRIGHT = 1
Same as ARW_BOTTOMRIGHT.

ARW_STARTTOP = 2
Same as ARW_TOPLEFT.

ARW_TOPLEFT = 2
Start placing the icons in the top-left corner of the screen.

ARW_TOPRIGHT = 3
Start placing the icons in the top-right corner of the screen.

ARW_UP = 0
Add new icons above existing ones.

SM_CLEANBOOT = 67

Win 95/98 only: Return a value specifying how the computer was booted up. 0 means a normal bootup, 1 means a fail-safe (a.k.a. SafeBoot) bootup, and 2 means a fail-safe bootup with the network installed.

SM_CMETRICS = 44

Win 95/98 only: Return the number of system metrics.
SM_CMOUSEBUTTONS = 43
    Return the number of mouse buttons on the installed mouse, or 0 if no mouse is installed.
SM_CXBORDER = 5
    Win NT: Return the width of a window border. Win 95/98: Return the width of a single window border.
SM_CXCURSOR = 13
    Win NT: Return the width of the cursor. Win 95/98: Return the width of the standard cursor bitmap.
SM_CXDIALOGFRAME = 7
    Win NT only: Return the width of a window frame having a dialog frame style.
SM_CXDOUBLECLK = 36
    The width of the rectangle within which both mouse clicks must be to recognize a double-click.
SM_CXDRA G = 68
    Return the minimum width the cursor must move to begin a drag-and-drop operation.
SM_CXEDGE = 45
    Win 95/98 only: Return the width of a 3D window border.
SM_CXFRAME = 32
    Win NT only: Return the width of the border of a resizable window.
SM_CXFULLSCREEN = 16
    Return the width of the client area of a full-screen window.
SM_CXHSCROLL = 21
    Win 95/98 only: Return the width of an arrow bitmap on a horizontal scrollbar.
SM_CXHTHUMB = 10
    Return the width of the horizontal scrollbar thumb.
SM_CXICON = 11
    Return the default width of an icon.
SM_CXICONSPACING = 38
    Win NT: Return the width of the cell used to position icons. Win 95/98: Return the width of the grid cell for items in a large icon view.
SM_CXMAXIMIZED = 61
    Win 95/98 only: Return the default width of a maximized window.
SM_CXMAXTRACK = 59
    Win 95/98 only: Return the default maximum width the user is allowed to resize a window to.
SM_CXMINUCHECK = 71
    Win 95/98 only: Return the width of the default menu check-mark bitmap.
SM_CXMENU SIZE = 54
    Win 95/98 only: Return the width of a menu bar button.
SM_CXMIN = 28
    Return the minimum width of a window.
SM_CXMINIMIZED = 57
    Win 95/98 only: Return the width of a normal minimized window.
SM_CXMINSPACING = 47
    Win 95/98 only: Return the width of the grid cell rectangle used to position minimized windows.
SM_CXMINTRACK = 34
    Win 95/98 only: Return the default minimum width the user is allowed to resize a window to.
SM_CXSCREEN = 0
    Return the width of the screen.
SM_CXSIZE = 30
    Win NT: Return the width of a title bar bitmap. Win 95/98: Return the width of a caption button.
SM_CXSIZEFRAME = 32
    Win 95/98 only: Return the width of a thick window frame.
SM_CXSICON = 49
    Win 95/98 only: Return the recommended width for small icons.
SM_CXSMSIZE = 52
  **Win 95/98 only:** Return the width of a small caption button.

SM_CXVSCROLL = 2
  **Win 95/98 only:** Return the width of a vertical scrollbar.

SM_CYBORDER = 6
  **Win NT:** Return the height of a window border. **Win 95/98:** Return the height of a single window border.

SM_CYCAPTION = 4
  Return the height of a normal caption area.

SM_CYCURSOR = 14
  **Win NT:** Return the height of the cursor. **Win 95/98:** Return the height of the standard cursor bitmap.

SM_CYDLGFRAME = 8
  **Win NT only:** Return the height of a window frame having a dialog frame style.

SM_CYDOUBLECLK = 37
  Return the height of the rectangle within which both mouse clicks must be to recognize a double-click.

SM_CYDRAG = 69
  Return the minimum height the cursor must move to begin a drag-and-drop operation.

SM_CYEDGE = 46
  **Win 95/98 only:** Return the height of a 3D window border.

SM_CYFRAME = 33
  **Win NT only:** Return the height of the border of a resizable window.

SM_CYFULLSCREEN = 17
  Return the height of the client area of a full-screen window.

SM_CYHSCROLL = 3
  **Win 95/98 only:** Return the height of a horizontal scrollbar.

SM_CYICON = 12
  Return the default height of an icon.

SM_CYICONSPACING = 39
  **Win NT:** Return the height of the cell used to position icons. **Win 95/98:** Return the height of the grid cell for items in a large icon view.

SM_CYKANJIWINDOW = 18
  Return the height of the Kanji window (for double-byte character set versions of Windows).

SM_CYMAXIMIZED = 62
  **Win 95/98 only:** Return the default height of a maximized window.

SM_CYMAXTRACK = 60
  **Win 95/98 only:** Return the default maximum height the user is allowed to resize a window to.

SM_CYMENU = 15
  Return the height of a single menu bar.

SM_CYMENUCHECK = 72
  **Win 95/98 only:** Return the height of the default menu check-mark bitmap.

SM_CYMENUSIZE = 55
  **Win 95/98 only:** Return the height of a menu bar button.

SM_CYMIN = 29
  Return the minimum height of a window.

SM_CYMINIMIZED = 58
  **Win 95/98 only:** Return the height of a normal minimized window.

SM_CYMINSPACING = 48
  **Win 95/98 only:** Return the height of the grid cell rectangle used to position minimized windows.

SM_CYMINTRACK = 35
  **Win 95/98 only:** Return the default minimum height the user is allowed to resize a window to.

SM_CYSCREEN = 1
  Return the height of the screen.
Windows API Guide: GetSystemMetrics Function

SM_CYSIZE = 31
   Win NT: Return the height of a title bar bitmap.  Win 95/98: Return the height of a caption button.
SM_CYSIZEFRAME = 33
   Win 95/98 only: Return the height of a thick window frame.
SM_CYSMCAPTION = 51
   Win 95/98 only: Return the height of a small caption area.
SM_CYSMICON = 50
   Win 95/98 only: Return the recommended height for small icons.
SM_CYSMSIZE = 53
   Win 95/98 only: Return the height of a small caption button.
SM_CYVSCROLL = 20
   Win 95/98 only: Return the height of an arrow bitmap on a vertical scrollbar.
SM_CYVTHUMB = 9
   Return the height of a vertical scrollbar thumb.
SM_DBCSENABLED = 42
   Return a non-zero value if the double-byte character set version of USER.EXE is installed, 0 if not.
SM_DEBUG = 22
   Return a non-zero value if the debugging version of USER.EXE is installed, 0 if not.
SM_MENUDROPALIGNMENT = 40
   Return a non-zero value if popup menus appear to the right, 0 if to the left.
SM_MIDEASTENABLED = 74
   Return a non-zero value if the system is enabled to use Hebrew and Arabic languages, 0 if not.
SM_MOUSEPRESENT = 19
   Return a non-zero value if a mouse is installed and present, 0 if not.
SM_NETWORK = 63
   Set the &H1 bit of the return value if a network is installed. All other bits of the return value are reserved and undefined.
SM_PENWINDOWS = 41
   Return a non-zero value if the Microsoft Windows for Pen computing extensions are installed, 0 if not.
SM_SECURE = 44
   Return a non-zero value if security is present and active, 0 if not.
SM_SHOWSOUNDS = 70
   Return a non-zero value if the application should show a visual cue for all sounds, 0 if not.
SM_SLOWMACHINE = 73
   Return a non-zero value if the system has a slow, low-end processor, 0 if not.
SM_SWAPBUTTON = 23
   Return a non-zero value if the left and right mouse buttons are swapped, 0 if not.

Example:

' Display the screen resolution (size) and some information about the
' configuration of windows.
Dim xres As Long, yres As Long  ' receive x and y resolutions of the display
Dim hasmouse As Long, numbuttons As Long  ' receive presence of mouse and number of
   buttons
Dim hasnetwork As Long  ' receives info on availability of network

' Display the screen resolution -- i.e., the width and height of the screen.
xres = GetSystemMetrics(SM_CXSCREEN)
yres = GetSystemMetrics(SM_CYSCREEN)
Debug.Print "The display is"; xres; "pixels wide and"; yres; "pixels high."
Display the number of buttons on the mouse, if a mouse is present.

```vbscript
hasmouse = GetSystemMetrics(SM_MOUSEPRESENT)
numbuttons = GetSystemMetrics(SM_CMOUSEBUTTONS)
If hasmouse = 0 Then
    Debug.Print "No mouse is installed on the system."
Else
    Debug.Print "A mouse with"; numbuttons; "buttons is installed."
End If
```

Display whether the system has a network connection installed.

```vbscript
hasnetwork = GetSystemMetrics(SM_NETWORK)
If (hasnetwork And &H1) = &H1 Then ' check only the information bit
    Debug.Print "A network is configured to be used by Windows."
Else
    Debug.Print "No network is currently configured."
End If
```

See Also: SystemParametersInfo

Category: Accessibility
GetSystemTime Function

Declare Sub GetSystemTime Lib "kernel32.dll" (lpSystemTime As SYSTEMTIME)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetSystemTime retrieves the current system date and time and places it into a SYSTEMTIME structure. Times are always reported in UTC time (Coordinated Universal Time, a.k.a. Greenwich Mean Time (GMT)), not in the system's local time.

Return Value

GetSystemTime does not return a value.

Visual Basic-Specific Issues

None.

Parameters

*lpSystemTime*

Receives the current system date and time.

Example
' Display the current time in UTC time (along the Prime Meridian).
Dim utctime As SYSTEMTIME ' receives time and date

GetSystemTime utctime ' put the time and date in UTC time into utctime
Debug.Print "In Greenwich, England, the time is"; utctime.wHour; ":"; utctime.wMinute; ":"; utctime.wSecond

See Also

GetLocalTime, GetSystemTimeAsFileTime, SetSystemTime

Category

Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetSystemTimeAsFileTime Function

Declare Sub GetSystemTimeAsFileTime Lib "kernel32.dll"
(lpSystemTimeAsFileTime As FILETIME)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Not Supported.

Description & Usage

GetSystemTimeAsFileTime retrieves the current system date and time and places it into a FILETIME structure. The system time is always reported in UTC time (Coordinated Universal Time, a.k.a. Greenwich Mean Time (GMT)).

Return Value

GetSystemTimeAsFileTime does not return a value.

Visual Basic-Specific Issues

None.

Parameters

lpSystemTimeAsFileTime
  Receives the current system date and time.

Example
This code is licensed according to the terms and conditions listed here.

Set the modification time of C:\MyApp\test.txt to the current system date and time. Leave the other times as they were before calling the function.

```
Dim hFile As Long ' handle to the opened file
Dim ctime As FILETIME ' the time of creation
Dim atime As FILETIME ' the time of last access
Dim mtime As FILETIME ' the time of last modification
Dim retval As Long ' return value

First, open the file C:\MyApp\test.txt for both read-level and write-level access, since we need to do both.

```

hFile = CreateFile("C:\MyApp\test.txt", GENERIC_READ Or GENERIC_WRITE, FILE_SHARE_READ, ByVal CLng(0), OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
```

If hFile = -1 Then
    Debug.Print "Could not open the file successfully -- aborting."
    End ' terminate the program
End If
```

Next, get the creation, last-access, and last-modification times.

```
retval = GetFileTime(hFile, ctime, atime, mtime)
```

Get the system time (already in UTC) as a FILETIME structure.

```
GetSystemTimeAsFileTime mtime
```

Set the retrieved creation and access times and the new modification time as the file's times.

```
retval = SetFileTime(hFile, ctime, atime, mtime)
```

Close the file to free up resources.

```
retval = CloseHandle(hFile)
```

See Also

GetSystemTime

Category

Time

Go back to the alphabetical Function listing.
GetTempFileName Function

Declare Function GetTempFileName Lib "kernel32.dll" Alias "GetTempFileNameA" (ByVal lpszPath As String, ByVal lpPrefixString As String, ByVal wUnique As Long, ByVal lpTempFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetTempFileName generates a filename for a temporary file and optionally creates it for you. Temporary files are used to store data for short periods of time on the hard drive. The full filename, including the path, is put into the string variable passed as lpTempFileName. The format of the generated filename is path\xxxuuuu.TMP. path is the path to put the file in, preferably the temporary directory supplied by Windows, which can be gotten through GetTempPath. xxx is the string specified in lpPrefixString. uuuu is a hexadecimal number between 0000 and FFFF. If wUnique is non-zero, uuuu is the rightmost four digits of the number (in hexadecimal), and the file is not created. This method will work even if a file with that name already exists. If wUnique is zero, uuuu is a random number generated by Windows, and the file is created. Temporary files always have a .TMP extension. The function returns the value used for uuuu if successful, or 0 if an error occurred.

lpszPath
The path to put the temporary file into. This is preferably the same path gotten from GetTempPath.

lpPrefixString
The first three characters of the filename.

wUnique
If non-zero, this number's last four digits in hexidecimal are the last four characters in the filename, and the file is not created. If zero, the last four characters are generated by Windows, and the file is created.

lpTempFileName
A string that receives the path and filename of the temporary file.

Example:

' Generate a temporary file (path)\api????.TMP, where (path) ' is Windows's temporary file directory and ???? is a randomly assigned unique value.
' Then display the name of the created file on the screen.
Dim temppath As String  ' receives name of temporary file path
Dim tempfile As String  ' receives name of temporary file
Dim slength As Long  ' receives length of string returned for the path
Dim lastfour As Long  ' receives hex value of the randomly assigned ????

' Get Windows's temporary file path
temppath = Space(255)  ' initialize the buffer to receive the path
slength = GetTempPath(255, temppath)  ' read the path name
temppath = Left(temppath, slength)  ' extract data from the variable

' Get a uniquely assigned random file
tempfile = Space(255)  ' initialize buffer to receive the filename
lastfour = GetTempFileName(temppath, "api", 0, tempfile)  ' get a unique temporary
file name
' (Note that the file is also created for you in this case.)
tempfile = Left(tempfile, InStr(tempfile, vbNullChar) - 1)  ' extract data from the
variable
Debug.Print "Temporary filename: "; tempfile

See Also: GetTempPath
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Go back to the Windows API Guide home page.
E-mail: rogue953@st-louis.crosswinds.net
This page is at http://www.vbapi.com/ref/g/gettempfilename.html
GetTempPath Function

Declare Function GetTempPath Lib "kernel32.dll" Alias "GetTempPathA" (ByVal nBufferLength As Long, ByVal lpBuffer As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetTempPath determines Windows's default Temp directory. The Temp directory is where temporary files made and used by Windows-based programs should be put. Usually this will be the \Temp subdirectory under the Windows directory, but not necessarily. The path of the directory is put into the string variable passed as lpBuffer. The function returns the length of the string returned by the function if it succeeds, or 0 if it fails.

nBufferLength
The length in characters of lpBuffer

lpBuffer
A string that will receive the path of the Temp directory.

Example:

' Generate a temporary file (path)\api????.TMP, where (path)
' is Windows's temporary file directory and ???? is a randomly assigned unique value.
' Then display the name of the created file on the screen.
Dim temppath As String ' receives name of temporary file path
Dim tempfile As String ' receives name of temporary file
Dim slength As Long ' receives length of string returned for the path
Dim lastfour As Long ' receives hex value of the randomly assigned ????

' Get Windows's temporary file path
temppath = Space(255) ' initialize the buffer to receive the path
slength = GetTempPath(255, temppath) ' read the path name
temppath = Left(temppath, slength) ' extract data from the variable

' Get a uniquely assigned random file
tempfile = Space(255) ' initialize buffer to receive the filename
lastfour = GetTempFileName(temppath, "api", 0, tempfile) ' get a unique temporary file name
' (Note that the file is also created for you in this case.)
tempfile = Left(tempfile, InStr(tempfile, vbNullChar) - 1) ' extract data from the variable
Debug.Print "Temporary filename: "; tempfile

See Also: GetTempFileName
Category: System Information
GetTextAlign Function

Declare Function GetTextAlign Lib "gdi32.dll" (ByVal hdc As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetTextAlign determines how a device displays a line of text relative to a given reference point. The reference point is the point used to specify where the device should display a line of text. The function returns exactly three of the following flags specifying where this reference point will be relative to the text (one flag specifies horizontal position, one specifies vertical position, one determines current point updating):

TA_BASELINE = 24
   The reference point will be on the baseline of the text.
TA_BOTTOM = 8
   The reference point will be on the bottom edge of the bounding rectangle of the text.
TA_CENTER = 6
   The reference point will be horizontally centered along the bounding rectangle of the text.
TA_LEFT = 0
   The reference point will be on the left edge of the bounding rectangle of the text.
TA_NOUPDATECP = 0
   Do not set the current point to the reference point.
TA_RIGHT = 2
   The reference point will be on the right edge of the bounding rectangle of the text.
TA_RTLREADING = 256
   Win 95/98 only: Display the text right-to-left (if the font is designed for right-to-left reading).
TA_TOP = 0
   The reference point will be on the top edge of the bounding rectangle of the text.
TA_UPDATECP = 1
   Set the current point to the reference point.

hdc
   The device context of the device to find the reference point settings of.

Example:

' Display whether window Form1 will display text left-justified, ' centered, or right-justified relative to a given reference point. Dim refpoint As Long ' receives reference point settings
refpoint = GetTextAlign(Form1.hDC) ' get the text alignment setting of the window If (refpoint And TA_RIGHT) = TA_RIGHT Then ' ref. point on right edge Debug.Print "Text will be displayed right-justified."
ElseIf (refpoint And TA_CENTER) = TA_CENTER Then ' ref. point horizontally centered Debug.Print "Text will be displayed centered horizontally."
Else ' assume ref. point on left edge
    Debug.Print "Text will be displayed left-justified."
End If

See Also: SetTextAlign, TextOut
Category: Fonts & Text

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetThreadLocale Function

Declare Function GetThreadLocale Lib "kernel32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetThreadLocale identifies the locale of the thread which called the function (i.e., your program's thread). This locale determines the default preferences for display of various data for the program and may be different than that of other threads.

Return Value

The function returns the identifier of the calling thread's locale.

Visual Basic-Specific Issues

None.

Parameters

None.

Example

' This code is licensed according to the terms and conditions listed here.

' Use a case-sensitive, string sort comparison method to alphabetically sort nine words. The sorting method simply compares each possible pair of words; if a pair is out of alphabetical order, they are switched.
Dim words(1 To 9) As String ' the words to sort
Dim tempstr As String ' buffer used to swap strings
Dim oc As Integer, ic As Integer ' counter variables
Dim compval As Long  ' result of comparison
Dim threadlocale As Long  ' locale ID of this thread

' Get the locale of this thread (i.e., of this program).
threadlocale = GetThreadLocale()

' Load the nine strings into the array.
words(1) = "can't"
words(2) = "cant"
words(3) = "cannot"
words(4) = "pants"
words(5) = "co-op"
words(6) = "coop"
words(7) = "Denver"
words(8) = "denver"
words(9) = "denver"

' Sort the strings, swapping any pairs which are out of order.
For oc = 1 To 8  ' first string of the pair
    For ic = oc + 1 To 9  ' second string of the pair
        ' Compare the two strings.
        compval = CompareString(threadlocale, SORT_STRINGSORT, words(oc), Len(words(oc)),
                                words(ic), Len(words(ic)))
        ' If words(oc) is greater, swap them.
        If compval = CSTR_GREATER_THAN Then
            tempstr = words(oc)
            words(oc) = words(ic)
            words(ic) = tempstr
        End If
    Next ic
Next oc

' Display the list of sorted words.
For oc = 1 To 9
    Debug.Print words(oc)
Next oc

See Also

SetThreadLocale

Category

National Language Support

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetTickCount Function

Declare Function GetTickCount Lib "kernel32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Unknown.

Description & Usage

GetTickCount determines how much time has elapsed since Windows was last started. The time is measured in milliseconds, although the actual resolution of the function's output depends on that of the system timer itself. Therefore, it may not be perfectly accurate to the millisecond. Because of the limitations of the 32-bit integer data type, the reported elapsed time wraps back to zero after about 49.7 days of continuous operation.

Return Value

The function returns the number of milliseconds that have passed since Windows was last started.

Visual Basic-Specific Issues

None.

Parameters

None.

Example

' This code is licensed according to the terms and conditions listed here.
' Determine about how much time it takes for Visual Basic to compute the square root of a large number. The elapsed time for that operation is displayed in seconds.
Dim starttime As Long  ' timer value before the calculation
Dim endtime As Long   ' timer value after the calculation
Dim result As Double  ' receives result of square root operation

' Find how much time has passed since Windows was started.
starttime = GetTickCount()
' Calculate the square root of a large number.
result = Sqr(54761)
' Find how much time has now passed since startup.
endtime = GetTickCount()
' The difference between starttime and endtime is the time it took ' to calculate the square root.
Debug.Print "The calculation took"; (endtime - starttime) / 1000; "seconds."

Category

Time

Back to the Function list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/gettickcount.html
GetTimeFormat Function

Declare Function GetTimeFormat Lib "kernel32.dll" Alias "GetTimeFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, lpTime As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpTimeStr As String, ByVal cchTime As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetTimeFormat formats a string to display a time according to a locale's settings. The time can be formatted using either a predefined format or a custom format specified in the parameter list. The string generated by this function can be used to present a more human-readable way to display a time.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the size in bytes of the string placed into the variable passed as lpTimeStr.

Visual Basic-Specific Issues

When passing 0 for the lpFormat parameter, the expression CLng(0) must be used.

Parameters

Locale

The identifier of the locale to use to format the string as necessary. If this is 0, the locale of the calling thread is used. This can also be one of the following flags specifying a default locale:

- LOCALE_SYSTEM_DEFAULT
  - The system's default locale.
- LOCALE_USER_DEFAULT
The user's default locale.

dwFlags
A combination of the following flags specifying how to format the time string. If a format string is passed as lpFormat, this parameter must be set to 0.

LOCALE_NOUSEROVERRIDE
Use the system's default time format for the specified locale, ignoring any changes to those defaults which the user may have selected.

LOCALE_USE_CP_ACP
Use the system's ANSI code page for string translation instead of the locale's code page.

TIME_NOMINUTESORSECONDS
Do not display minutes or seconds.

TIME_NOSECONDS
Do not display seconds.

TIME_NOTIMEMARKER
Do not display an AM/PM marker.

TIME_FORCE24HOURFORMAT
Use a 24-hour time format.

lpTime
The time to format as a string. The members of the structure which specify the date are ignored.

lpFormat
The format template string used to generate the time string. To use one of the predefined formats, this parameter must be 0. In a format template string, the following series of characters stand for the following components of the time:

h
Hours without a leading zero for single-digit hours, using a 12-hour clock.

hh
Hours with a leading zero for single-digit hours, using a 12-hour clock.

H
Hours without a leading zero for single-digit hours, using a 24-hour clock.

HH
Hours with a leading zero for single-digit hours, using a 24-hour clock.

m
Minutes without a leading zero for single-digit minutes.

mm
Minutes with a leading zero for single-digit minutes.

s
Seconds without a leading zero for single-digit seconds.

ss
Seconds with a leading zero for single-digit seconds.

t
A one-character AM/PM indicator.

tt
A two-character AM/PM indicator.

Any spaces appearing in the template string appear verbatim in the formatted string. To place any other "fixed" characters or text in the format string, you must enclose the literal text in single quotation marks.

lpTimeStr
Receives the formatted time string. This must initially be sufficiently long to receive the string.

cchTime
The length of the string passed as lpTimeStr.
Constant Definitions

Const LOCALE_SYSTEM_DEFAULT = &H400
Const LOCALE_USER_DEFAULT = &H800
Const LOCALE_NOUSEROVERRIDE = &H80000000
Const LOCALE_USE_CP_ACP = &H40000000
Const TIME_NOMINUTESORSECONDS = &H1
Const TIME_NOSECONDS = &H2
Const TIME_NOTIMEMARKER = &H4
Const TIME_FORCE24HOURFORMAT = &H8

Example

' This code is licensed according to the terms and conditions listed here.

' Display today's time first using a "regular" hours-minutes-
' AM/PM string and then using a "military hours" format (e.g., 8:27 AM is 0827
' hours).
Dim today As SYSTEMTIME  ' today's date and time
Dim timestr As String  ' receives the formatted time string
Dim strlen As Long  ' length of the buffer for the formatted time string

' Get today's date and time in the local time zone.
GetLocalTime today

' Make sufficient room in the buffer to receive the time string.
timestr = Space(255)
' Format today's time as hours-minutes-AM/PM.
strlen = GetTimeFormat(0, TIME_NOSECONDS, today, CLng(0), timestr, Len(timestr))
' Remove the empty space from the formatted time string.
timestr = Left(timestr, strlen)
' Display today's time using that format.
Debug.Print "It is currently "; timestr

' Now make sufficient room once again.
timestr = Space(255)
' Format today's time in the military-esque format.
strlen = GetTimeFormat(0, 0, today, "HHmm 'hours'", timestr, Len(timestr))
' Remove the empty space from the formatted string.
timestr = Left(timestr, strlen)
' Display today's time in aforementioned format.
Debug.Print "It is currently "; timestr

See Also

GetDateFormat
GetTimeZoneInformation Function

Declare Function GetTimeZoneInformation Lib "kernel32.dll" (lpTimeZoneInformation As TIME_ZONE_INFORMATION) As Long

Platforms: Win 95/98, Win NT

GetTimeZoneInformation reads the computer's current time zone settings. Since Windows handles all of the system clock settings, there usually isn't a need for other programs to use this information. The information is put into the variable passed as lpTimeZoneInformation. The function returns 0 if an error occurred, or a 1 if successful.

lpTimeZoneInformation

Variable which receives the information about the system time zone.

Example:

' Display the name of the time zone the computer is set to.
Dim tzi As TIME_ZONE_INFORMATION  ' receives information on the time zone
Dim retval As Long  ' return value
Dim c As Long  ' counter variable needed to display time zone name

retval = GetTimeZoneInformation(tzi)  ' read information on the computer's selected time zone

' Oddly, instead of being stored in a string, the time zone name is stored in a ' 32-element array, each element holding the ASCII code of one of the characters. ' This loop converts the array into a readable string.
Debug.Print "The computer's time zone is: ";
For c = 0 To 31  ' the array's range is from 0 to 31
  If tzi.StandardName(c) = 0 Then Exit For  ' abort if the terminating null character is reached
  Debug.Print Chr(tzi.StandardName(c))  ' convert the ASCII code into a character and display it
Next c
Debug.Print ";"  ' end the line being displayed

Category: Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
GetTopWindow Function

Declare Function GetTopWindow Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetTopWindow returns a handle to the currently active child window of a window. The active child window is the one that has the focus, and it is usually at the top of all the other children in the Z-order. This function works even if the parent window is not active. If an error occurs or the window has no children, the function instead returns 0. This function is identical to calling GetWindow using the GW_CHILD relationship.

hwnd
The handle of the parent window. The function will return its active child window.

Example:

' Flash the MDI form window MDIForm1's active child once.
' (In VB, a MDI form has child windows).
Dim active As Long ' receives handle to the MDI form's active window
Dim retval As Long ' return value used for flashing the child window
active = GetTopWindow(MDIForm1.hWnd) ' get the handle of MDIForm1's active child window
If active <> 0 Then ' don't try to flash if there is no child window
    ' The next three lines flags the window once.
    retval = FlashWindow(active, 1): Sleep 250
    retval = FlashWindow(active, 1): Sleep 250
    retval = FlashWindow(active, 0)
End If

See Also: GetWindow
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/gettopwindow.html
GetUserName Function

Declare Function GetUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, nSize As Long) As Long

Platforms: Win 95/98, Win NT

GetUserName retrieves the name of the user who is logged on to Windows. This user's saved settings are the ones used during the current Windows session. The name of the user is placed in the string passed as lpBuffer. The function puts the size of the returned string into the variable passed as nSize. The function returns 0 if an error occurred, or a non-zero value if successful.

lpBuffer
String which receives the name of the user logged on to Windows. The string passed must have sufficient room to receive the string the function will give it.

nSize
The size of the string passed to the function. The variable passed as this parameter also receives the size of the returned string (including the terminating null character).

Example:

' Display the name of the user currently logged on.
Dim username As String ' receives name of the user
Dim slength As Long ' length of the string
Dim retval As Long ' return value

' Create room in the buffer to receive the returned string.
username = Space(255) ' room for 255 characters
slength = 255 ' initialize the size of the string
' Get the user's name and display it.
retval = GetUserName (username, slength) ' slength is now the length of the returned string
username = Left(username, slength - 1) ' extract the returned info from the buffer
' (We subtracted one because we don't want the null character in the trimmed string.)
Debug.Print "The name of the current user is "; username

Category: System Information

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
**GetVersionEx Function**

Declare Function GetVersionEx Lib "kernel32.dll" Alias "GetVersionExA"
(lpVersionInformation As OSVERSIONINFO) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

*GetVersionEx* reads information about the version of Windows running as the operating system. This information includes the strict version number and platform (3.x with Win32s, Windows 95, Windows NT, Windows 98, etc.). The information is put into the variable passed as *lpVersionInformation*. The function returns 0 if an error occurred, or a 1 if successful.

*lpVersionInformation*

- Receives the version information for the operating system.

**Example:**

' Display the major and minor version numbers of Windows.
' For example, 4.0 could represent Windows 95.
Dim os As OSVERSIONINFO ' receives version information
Dim retval As Long ' return value

os.dwOSVersionInfoSize = Len(os) ' set the size of the structure
retval = GetVersionEx(os) ' read Windows's version information
Debug.Print "Windows version number:"; os.dwMajorVersion; ";"; os.dwMinorVersion

**Category:** System Information

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetVolumeInformation Function

Declare Function GetVolumeInformation Lib "kernel32.dll" Alias "GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long, lpVolumeSerialNumber As Long, lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

GetVolumeInformation retrieves information describing a disk volume and the file system it uses. This information includes things such as the volume label and the disk's serial number.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- lpRootPathName
  The root directory of the disk volume to be described. This directory must include a trailing backslash character. For example, to describe the C: drive, this would be "C:\".

- lpVolumeNameBuffer
  Receives the null-terminated volume label of the volume. This string must be large enough to receive the text.

- nVolumeNameSize
  The length of the string passed as lpVolumeNameBuffer.

- lpVolumeSerialNumber
- lpFileSystemFlags
- lpFileSystemNameBuffer
- nFileSystemNameSize
Receives the serial number of the volume. When displayed, this value should be shown in hexadecimal with a dash before the last four hex digits. **Windows 95/98:** If the volume is a network drive, the serial number will not be retrieved.

`lpMaximumComponentLength`

Receives the maximum length of a file name component supported by the file system. A file name component is any portion of a file name between backslashes. This value is used primarily to determine if the volume supports long file names or if it is limited to the old 8.3 system.

`lpFileSystemFlags`

A combination of the following flags describing other features of the file system used by the volume:

- **FS_CASE_IS_PRESERVED**
  The file system preserves the case of file names.
- **FS_CASE_SENSITIVE**
  The file system uses case-sensitive file names.
- **FS_UNICODE_STORED_ON_DISK**
  The file system supports Unicode in file names as they appear on disk.
- **FS_PERSISTENT_ACLS**
  The file system preserves and enforces access control lists (ACLs).
- **FS_FILE_COMPRESSION**
  The file system supports file-based compression, where individual files can be compressed while others are not.
- **FS_VOL_IS_COMPRESSED**
  The entire volume is compressed; for example, DoubleSpace has been used on the disk.
- **FILE_NAMED_STREAMS**
  The file system supports named streams.
- **FILE_SUPPORTS_ENCRYPTION**
  The file system supports the Encrypted File System (EFS).
- **FILE_SUPPORTS_OBJECT_IDS**
  The file system supports object identifiers.
- **FILE_SUPPORTS_REPARSE_POINTS**
  The file system supports reparse points.
- **FILE_SUPPORTS_SPARSE_FILES**
  The file system supports sparse files.
- **FILE_VOLUME_QUOTAS**
  The file system supports disk quotas.

`lpFileSystemNameBuffer`

Receives the null-terminated name of the file system. This string must be long enough to receive the text.

`nFileSystemNameSize`

The length of the string passed as `lpFileSystemNameBuffer`.

### Constant Definitions

```plaintext
Const FS_CASE_IS_PRESERVED = &H2
Const FS_CASE_SENSITIVE = &H1
Const FS_UNICODE_STORED_ON_DISK = &H4
Const FS_PERSISTENT_ACLS = &H8
Const FS_FILE_COMPRESSION = &H10
Const FS_VOL_IS_COMPRESSED = &H8000
Const FILE_NAMED_STREAMS = &H40000
Const FILE_SUPPORTS_ENCRYPTION = &H20000
Const FILE_SUPPORTS_OBJECT_IDS = &H10000
```
Const FILE_SUPPORTS_REPARSE_POINTS = &H80
Const FILE_SUPPORTS_SPARSE_FILES = &H40
Const FILE_VOLUME_QUOTAS = &H20

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetVolumeInformation Lib "kernel32.dll" Alias _
    "GetVolumeInformationA" (ByVal lpRootPathName As String, ByVal _
    lpVolumeNameBuffer As String, ByVal nVolumeNameSize As Long, lpVolumeSerialNumber As Long, _
    lpMaximumComponentLength As Long, lpFileSystemFlags As Long, ByVal _
    lpFileSystemNameBuffer As String, ByVal nFileSystemNameSize As Long) As Long

' Display the volume label, serial number, and file system name
' of the C: drive.  Note how the serial number value is manipulated to
' display it properly.
Dim volname As String   ' receives volume name of C:
Dim sn As Long          ' receives serial number of C:
Dim snstr As String     ' display form of serial number
Dim maxcomplen As Long  ' receives maximum component length
Dim sysflags As Long    ' receives file system flags
Dim sysname As String   ' receives the file system name
Dim retval As Long      ' return value

' Initialize string buffers.
volname = Space(256)
sysname = Space(256)
' Get information about the C: drive's volume.
retval = GetVolumeInformation("C:\", volname, Len(volname), sn, maxcomplen, _
    sysflags, sysname, Len(sysname))
' Remove the trailing nulls from the two strings.
volname = Left(volname, InStr(volname, vbNullChar) - 1)
sysname = Left(sysname, InStr(sysname, vbNullChar) - 1)
' Format the serial number properly.
snstr = Trim(Hex(sn))
snstr = String(8 - Len(snstr), "0") & snstr
snstr = Left(snstr, 4) & ":" & Right(snstr, 4)
' Display the volume name, serial number, and file system name.
Debug.Print "Volume Name: "; volname
Debug.Print "Serial Number: "; snstr
Debug.Print "File System: "; sysname

See Also

SetVolumeLabel
GetWindow Function

Declare Function GetWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal wCmd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetWindow returns the handle of a window related to a given window. The relations generally deal with child-parent relationships or relationships among children of the same parent window. The exact relation is specified by wCmd. If an error occurs or there is no window having the specified relation, the function instead returns 0.

hwnd
  The handle of the first window in the relation.

wCmd
  Exactly one of the following flags specifying which relationship the returned window has to the given window:
  GW_HWNDFIRST = 0
    The highest window in the Z-order having the same parent as the given window.
  GW_HWNDLAST = 1
    The lowest window in the Z-order having the same parent as the given window.
  GW_HWNDNEXT = 2
    The window below the given window in the Z-order.
  GW_HWNDPREV = 3
    The window above the given window in the Z-order.
  GW_OWNER = 4
    The window that owns the given window (not to be confused with the parent window).
  GW_CHILD = 5
    The topmost of the given window's child windows. This has the same effect as using the GetTopWindow function.

Example:

' Flash the application's window that is below Form1 in the Z-order once.
Dim next As Long  ' receives handle of next window in the Z-order
Dim retval As Long  ' return value for flashing the window

next = GetWindow(Form1.hWnd, GW_HWNDNEXT)  ' get the handle of the next window
If next <> 0 Then  ' don't try to flags if no such window exists
   ' The next three lines flags the window once.

retval = FlashWindow(next, 1): Sleep 250
retval = FlashWindow(next, 1): Sleep 250
retval = FlashWindow(next, 0)
End If

See Also: GetActiveWindow, GetTopWindow
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GetWindowLong Function

Declare Function GetWindowLong Lib "user32.dll" Alias "GetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetWindowLong retrieves a 32-bit value from the information about a window. This function can also read a 32-bit value from the block of extra memory given to the window, if one exists.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the 32-bit value which was retrieved.

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to the window to retrieve a 32-bit value from.

nIndex
To get a 32-bit value from the window's extra memory block, this is the zero-based offset of the byte to begin reading from. Valid values range from 0 to the size of the extra memory block in bytes minus four. To get a 32-bit value from the properties of the window, this is one of the following flags specifying which piece of information to retrieve:
- GWL_EXSTYLE
  Retrieve the extended window styles of the window.
- GWL_HINSTANCE
  Retrieve a handle to the owning application's instance.
GWL_HWNDPARENT
  Retrieve a handle to the parent window, if any.
GWL_ID
  Retrieve the identifier of the window.
GWL_USERDATA
  Retrieve the application-defined 32-bit value associated with the window.
GWL STYLE
  Retrieve the window styles of the window.
GWL WNDPROC
  Retrieve a pointer to the WindowProc hook function acting as the window's procedure.
If the window happens to be a dialog box, this could also be one of the following flags:
DWL DLGPROC
  Retrieve a handle to the WindowProc hook function acting as the dialog box procedure.
DWL_MSGRESULT
  Retrieve the return value of the last message processed by the dialog box.
DWL_USER
  Retrieve the application-defined 32-bit value associated with the dialog box.

**Constant Definitions**

Const GWL_EXSTYLE = -20
Const GWL_HINSTANCE = -6
Const GWL_HWNDPARENT = -8
Const GWL ID = -12
Const GWL_STYLE = -16
Const GWL_USERDATA = -21
Const GWL WNDPROC = -4
Const DWL DLGPROC = 4
Const DWL_MSGRESULT = 0
Const DWL_USER = 8

**Example**

When the user clicks button Command1, determine if its parent window has a maximize button or not by checking its window style. To use this example, you must first place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetWindowLong Lib "user32.dll" Alias "GetWindowLongA" (ByVal hWnd As Long, _
  ByVal nIndex As Long) As Long
Public Const GWL_STYLE = -16
Public Const WS_MAXIMIZEBOX = &H10000

' *** Place the following code inside a form window. ***

Private Sub Command1_Click ()
Dim styles As Long  ' receives window styles of Form1

' Get the window styles of Form1.
styles = GetWindowLong(Me.hWnd, GWL_STYLE)
' Determine if a maximize box exists or not.
If (styles And WS_MAXIMIZEBOX) = WS_MAXIMIZEBOX Then
    Debug.Print "The form window has a maximize box."
Else
    Debug.Print "The form window does not have a maximize box."
End If

End Sub

See Also

GetClassLong, SetWindowLong

Category

Window Classes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getwindowlong.html
GetWindowRect Function

Declare Function GetWindowRect Lib "user32.dll" (ByVal hwnd As Long, lpRect As RECT) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

GetWindowRect reads the size and position of a window. This information is put into the variable passed as *lpRect*. The rectangle receives the coordinates of the upper-left and lower-right corners of the window. If the window is past one of the edges of the screen, the values will reflect that (for example, if the left edge of a window is off the screen, the rectangle's .Left property will be negative). The function returns 0 if an error occurred, or 1 if successful.

hwnd
The handle of the window to read the position and width of.

lpRect
Variable that receives the coordinates of the upper-left and lower-right corners of the window.

**Example:**

' Display the width and height of window Form1
' Width and height can be calculated from the coordinates returned in the rectangle.
Dim r As RECT ' receives window rectangle
Dim retval As Long ' return value

retval = GetWindowRect(Form1.hWnd, r) ' set r equal to Form1's rectangle
Debug.Print "Width ="; r.Right - r.Left
Debug.Print "Height ="; r.Bottom - r.Top

**See Also:** MoveWindow, SetWindowPos

**Category:** Windows

Go back to the alphabetical Function listing.  
Go back to the Reference section index.
GetWindowsDirectory Function

Declare Function GetWindowsDirectory Lib "kernel32.dll" Alias "GetWindowsDirectoryA" (ByVal lpBuffer As String, ByVal nSize As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetWindowsDirectory reads the path of the Windows directory. This is where Windows itself is stored, along with all of the little applets that come with it. Never assume this is "C:Windows" because it doesn't necessarily have to be there. The directory is put into the string variable passed as lpBuffer. The function returns the length of the returned string if successful, or 0 if an error occurred.

lpBuffer
String variable that receives the path.

nSize
The length in characters of lpBuffer.

Example:

' Display the location of the Windows directory
Dim windir As String  ' receives path of Windows directory
Dim slength As Long  ' receives length of the string returned

windir = Space(255)  ' initialize buffer to receive the string
slength = GetWindowsDirectory(windir, 255)  ' read the path of the Windows directory
windir = Left(windir, slength)  ' extract the returned string from the buffer
Debug.Print "The Windows directory is at: "; windir

See Also: GetSystemDirectory
Category: System Information
GetWindowRgn Function

Declare Function GetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.51 or later.
- Windows CE: Not Supported.

Description & Usage

GetWindowRgn obtains a copy of a window's current region. If a non-empty region is applied to a window, it restricts the area of the window that is visible, effectively changing its shape as far as the user is concerned. This function copies a window's region into a second region passed to the function.

Return Value

The function returns one of the following flags specifying the result of the region combination operation:

- ERROR
  - An error occurred while trying to combine the regions.
- NULLREGION
  - The combined region is empty, i.e., null.
- SIMPLEREGION
  - The combined region forms a rectangle.
- COMPLEXREGION
  - The combined region is not empty but is also not a rectangle.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to get the region of.
**hRgn**

A handle to a region that receives a copy of the window's region. It doesn't matter what the region passed to the function happens to be, because it is overwritten by the function.

**Constant Definitions**

```
Const ERROR = 0
Const NULLREGION = 1
Const SIMPLEREGION = 2
Const COMPLEXREGION = 3
```

**Example**

Use an elliptic region to make window Form1 appear elliptical. Notice how, as this example is written, a portion of the title bar is still visible after applying the region. This allows us to move the window without adding any special code providing a different way for the user to move the window. Normally, in a real program, you would not want the title bar to be displayed, but after all, this is just an example. At least it shows you that the region only changes what part of the window you can see -- it doesn't change anything else about the window.

To run this example, make two windows Form1 and Form2. The latter will be used to illustrate **GetWindowRgn**. Place the following three buttons on Form1: a button labeled "Apply Region" and named cmdApplyRegion, a button labeled 'Remove Region' and named cmdRemoveRegion, and a button labeled 'Show Region' and named cmdShowRegion. Place these buttons near the center of Form1 to make sure parts of them won’t be hidden when the region is applied.

'This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getwindowrgn.html).

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Declare Function GetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long) As Long
Public Declare Function SetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long, ByVal bRedraw As Boolean) As Long
Public Declare Function CreateEllipticRgn Lib "gdi32.dll" (ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long) As Long
Public Declare Function DeleteObject Lib "gdi32.dll" (ByVal hObject As Long) As Long
Public Declare Function InvertRgn Lib "gdi32.dll" (ByVal hdc As Long, ByVal hrgn As Long) As Long

' *** Place the following code inside Form1. ***

'Stores a handle to the special window region, if it exists.
Private hRgnWindow As Long

Private Sub Form_Load()
  Form2.Show
End Sub
' Initially disable the "Remove Region" button, since one hasn't
' yet been applied.
   cmdRemoveRegion.Enabled = False
End Sub

Private Sub cmdApplyRegion_Click()
' Create an elliptic region slightly smaller than the current size of
' Form1, and make that the window region. This makes the previously
' rectangular Form1 appear to be an ellipse.

   Dim retval As Long ' return value

   ' First, make the elliptical region, slightly smaller than Form1.
   hRgnWindow = CreateEllipticRgn(5, 5, (Form1.Width / Screen.TwipsPerPixelX) -
   5, _
   (Form1.Height / Screen.TwipsPerPixelY) - 5)

   ' Apply this region to Form1 and show the change immediately.
   retval = SetWindowRgn(Form1.hWnd, hRgnWindow, True)

   ' To make sure that multiple regions aren't created, disable
   ' this button and enable the "Remove Region" button.
   cmdApplyRegion.Enabled = False
   cmdRemoveRegion.Enabled = True

End Sub

Private Sub cmdRemoveRegion_Click()
' Remove the window region from Form1, returning it to its
' normal rectangular shape.

   Dim retval As Long ' return value

   ' Set a null window region, which removes the current one entirely.
   retval = SetWindowRgn(Form1.hWnd, 0, True)

   ' Delete the region object because it is no longer needed.
   retval = DeleteObject(hRgnWindow)

   ' Since the region no longer exists, enable "Apply Region" and
   ' disable this button.
   cmdApplyRegion.Enabled = True
   cmdRemoveRegion.Enabled = False

End Sub

Private Sub cmdShowRegion_Click()
' Show the region currently applied to Form1 by inverting that region on
' Form2. Note that if no region is applied, nothing appears to happen
' because
' the actual window region is empty.

   Dim hRgnCopy As Long ' region that receives copy of Form1's region
   Dim retval As Long ' return value

   ' Create a region. It doesn't matter what, since it will be overwritten
   ' when GetWindowRgn is called. We just need to have some region.
hRgnCopy = CreateEllipticRgn(0, 0, 0, 0)
' Copy Form1's region to hRgnCopy. The actual value of the handle
does not change, but the information it "points" to does.
retval = GetWindowRgn(Form1.hWnd, hRgnCopy)
' Invert the colors on Form2 that lie within this region.
retval = InvertRgn(Form2.hDC, hRgnCopy)
' Delete the copied region, since we no longer need it.
retval = DeleteObject(hRgnCopy)

End Sub

See Also

SetWindowRgn

Category

Painting & Drawing

Back to the Function list.
Back to the Reference section.

---

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getwindowrgn.html
GetWindowText Function

Declare Function GetWindowText Lib "user32.dll" Alias "GetWindowTextA" (ByVal hWnd As Long, ByVal lpString As String, ByVal nMaxCount As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

GetWindowText retrieves the text of a window. For regular windows, this is the text which appears in the title bar. For controls, this is the text in the control. Note that GetWindowText cannot retrieve the text in a control owned by another program. To get that text, use the WM_GETTEXT message instead.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of characters copied into the string passed as lpString, not counting the terminating null character.

Visual Basic-Specific Issues

None.

Parameters

hWnd
Windows API Guide: GetWindowText Function

A **handle** to the window to read the title of.

**lpString**
String variable that receives the window's text. This must already have enough room to receive the string.

**nMaxCount**
The length in characters of **lpString**.

**Example**

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/g/getwindowtext.html).

' Display the text displayed in the title bar of window Form1
Dim textlen As Long  ' receives length of text of the window
Dim wintext As String  ' receives the text of the window
Dim slength As Long  ' receives the length of the returned string

' Find out how many characters are in the window's text.
' Add 1 to compensate for the terminating null.
textlen = GetWindowTextLength(Form1.hWnd) + 1
' Make sufficient room in the buffer.
wintext = Space(textlen)
' Retrieve the text of window Form1.
slength = GetWindowText(Form1.hWnd, wintext, textlen)
' Remove the empty space from the string, if any.
wintext = Left(wintext, slength)
' Display the result.
Debug.Print "The title bar of window Form1 is: "; wintext

**See Also**

GetWindowTextLength, SetWindowText, WM_GETTEXT

**Category**

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** February 11, 2000
GetWindowTextLength Function

Declare Function GetWindowTextLength Lib "user32.dll" Alias "GetWindowTextLengthA" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

GetWindowTextLength returns the length in characters of a window's text. You can use this function in conjunction with GetWindowText to create a string just long enough to receive the text. However, this function does not include the terminating null character in the window's text in the character count. In some instances, this function might report a larger text length than actually exists; however, it will never report fewer than the actual number of characters. GetWindowTextLength does not work with controls owned by other programs. To get the window text length of these controls, use the WM_GETTEXTLENGTH message instead.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of characters in the window's text, not including the terminating null character.

Visual Basic-Specific Issues

None.
Parameters

*hWnd*

A handle to the window to retrieve the length of the window text of.

Example

' Display the text displayed in the title bar of window Form1
Dim textlen As Long ' receives length of text of the window
Dim wintext As String ' receives the text of the window
Dim slength As Long ' receives the length of the returned string

' Find out how many characters are in the window's text.
' Add 1 to compensate for the terminating null.
textlen = GetWindowTextLength(Form1.hWnd) + 1
' Make sufficient room in the buffer.
wintext = Space(textlen)
' Retrieve the text of window Form1.
slength = GetWindowText(Form1.hWnd, wintext, textlen)
' Remove the empty space from the string, if any.
wintext = Left(wintext, slength)
' Display the result.
Debug.Print "The title bar of window Form1 is: "; wintext

See Also

GetWindowText, WM_GETTEXTLENGTH

Category

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

GetWindowThreadProcessId Function

Declare Function GetWindowThreadProcessId Lib "user32.dll" (ByVal hwnd As Long, lpdwProcessId As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GetWindowThreadProcessId finds identifiers for the thread which owns a given window and the process which created it. These identifiers can be used to later get information about the program controlling the window. Note that these two values are not handles but just numerical identifiers. The process identifier is put into the variable passed as lpdwProcessId, while the function returns the thread identifier.

hwnd
A handle to the window to find the identifiers of the owning thread and creating process.

lpdwProcessId
Receives the identifier for the process which created the window.

Example:

' Display the title bar text of all windows controlled by the thread ' which the window Form1 is in. This task is given to the callback function, which ' will receive each handle individually. Note that if the window has no title bar ' text, it will not be displayed (for clarity's sake).

' *** Place this code in a module. This is the callback function. ***
' This function displays the title bar text of the window identified by hwnd.
Public Function EnumThreadWndProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
    Dim slength As Long, wintext As String  ' title bar text length and buffer
    Dim retval As Long  ' return value
    Static winnum As Integer  ' counter keeps track of how many windows have been enumerated
    winnum = winnum + 1  ' one more window enumerated....
    slength = GetWindowTextLength(hwnd) + 1  ' get length of title bar text
    If slength > 1  ' if return value refers to non-empty string
        buffer = Space(slength)  ' make room in the buffer
        retval = GetWindowText(hwnd, buffer, slength)  ' get title bar text
        Debug.Print "Window "; winnum; "; " ; wintext:  ' display number of enumerated window
        Debug.Print Left(buffer, slength - 1)  ' display title bar text of enumerated window
    End If
    EnumThreadWndProc = 1  ' return value of 1 means continue enumeration
End Function
' *** Place this code wherever you want to enumerate the windows. ***
Dim threadid As Long, processid As Long  ' receive id to thread and process of Form1
Dim retval As Long  ' return value

' Determine the thread which owns the window Form1.
threadid = GetWindowThreadProcessId(Form1.hWnd, processid)
' Use the callback function to list all of the enumerated thread windows.  Note that
' lParam
' is set to 0 because we don't need to pass any additional information to the
function.
retval = EnumThreadWindows(threadid, AddressOf EnumThreadWndProc, 0)

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/getwindowthreadprocessid.html
GlobalAlloc Function

Declare Function GlobalAlloc Lib "kernel32.dll" (ByVal wFlags As Long, ByVal dwBytes As Long) As Long

**Platforms:** Win 32s, Win 95/98, Win NT

**GlobalAlloc** allocates a series of bytes in the computer's global memory. The memory can be used for any purpose necessary. The function's return value, if successful, depends on the flags specified in **wFlags**. It will either be a pointer to the block of allocated memory or a handle to the block of allocated memory. Although they both identify the same thing, they will most likely not be equal and cannot be used interchangably! If the function fails, a value of 0 will be returned. Note that Windows will not allocate a memory block starting at base address 0.

**wFlags**

One or more of the following flags specifying how to allocate the block of memory:

- GHND = &H40  
  Same as combining GMEM_MOVEABLE with GMEM_ZEROINIT.
- GMEM_DDESHARE = &H2000  
  Optimize the allocated memory for use in DDE conversations.
- GMEM_DISCARDABLE = &H100  
  Allocate discardable memory. (Cannot be combined with GMEM_FIXED.)
- GMEM_FIXED = &H0  
  Allocate fixed memory. The function's return value is a pointer to the beginning of the memory block. (Cannot be combined with GMEM_DISCARDABLE or GMEM_MOVEABLE.)
- GMEM_MOVEABLE = &H2  
  Allocate moveable memory. The memory block's lock count is initialized at 0 (unlocked). The function's return value is a handle to the beginning of the memory block. (Cannot be combined with GMEM_FIXED.)
- GMEM_NOCOMPACT = &H10  
  Do not compact any memory or discard any discardable memory to allocate the requested block.
- GMEM_NODISCARD = &H20  
  Do not discard any discardable memory to allocate the requested block.
- GMEM_SHARE = &H2000  
  Same as GMEM_DDESHARE.
- GMEM_ZEROINIT = &H40  
  Initialize the contents of the memory block to 0.
- GPTR = &H42  
  Same as combining GMEM_FIXED with GMEM_ZEROINIT.

**dwBytes**

The number of bytes to allocate; i.e., the size of the memory block to allocate.

**Example:**
' Use a block of memory as an intermediary step to copy
' the contents of array s() to array t(). Yes, you could copy them directly,
' but this demonstrates a few different memory functions.
Dim s(0 To 255) As Integer, t(0 To 255) As Integer ' arrays to copy from/to
Dim c As Integer, retval As Long ' counter variable & return value
Dim hMem As Long, pMem As Long ' handle and pointer to memory block

' Initialize the source array s()'s data
For c = 0 To 255
    s(c) = 2 * c ' each element equals double its index
Next c

' Allocate a moveable block of memory (returns a handle) (Integer type = 2 bytes)
hMem = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, 256 * 2)
' Lock the memory block, returning a pointer to it
pMem = GlobalLock(hMem)
' Copy the entire contents of s() to the memory block
Note that pMem is ByVal because we want its contents, not a pointer to it
CopyMemory ByVal pMem, s(0), 256 * 2
' Copy the contents of the memory block to t() (we could have just copied s() to t())
CopyMemory t(0), ByVal pMem, 256 * 2
' Unlock the memory block, destroying the pointer and freeing resources
x = GlobalUnlock(hMem)
' Free the memory block (de-allocate it)
x = GlobalFree(hMem)

' Verify that t() = s(), which it should
For c = 0 To 255
    If s(c) <> t(c) Then Debug.Print "Copy attempt failed."
End If

See Also: GlobalFree
Category: Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GlobalFree Function

Declare Function GlobalFree Lib "kernel32.dll" (ByVal hMem As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GlobalFree frees up the resources associated with a memory block, including of course the memory itself. This function should be used to deallocate any memory blocks allocated by GlobalAlloc after you are finished using it. The function returns 0 if successful, or the value passed as hMem if an error occurred.

hMem
   The handle to the block of memory to free.

Example:

' Use a block of memory as an intermediary step to copy
' the contents of array s() to array t(). Yes, you could copy them directly,
' but this demonstrates a few different memory functions.
Dim s(0 To 255) As Integer, t(0 To 255) As Integer ' arrays to copy from/to
Dim c As Integer, retval As Long ' counter variable & return value
Dim hMem As Long, pMem As Long ' handle and pointer to memory block

' Initialize the source array s()'s data
For c = 0 To 255
    s(c) = 2 * c ' each element equals double its index
Next c

' Allocate a moveable block of memory (returns a handle) (Integer type = 2 bytes)
hMem = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, 256 * 2)
' Lock the memory block, returning a pointer to it
pMem = GlobalLock(hMem)
' Copy the entire contents of s() to the memory block
' Note that pMem is ByVal because we want its contents, not a pointer to it
CopyMemory ByVal pMem, s(0), 256 * 2
' Copy the contents of the memory block to t() (we could have just copied s() to t())
CopyMemory t(0), ByVal pMem, 256 * 2
' Unlock the memory block, destroying the pointer and freeing resources
x = GlobalUnlock(hMem)
' Free the memory block (de-allocate it)
x = GlobalFree(hMem)
' Verify that t() = s(), which it should
For c = 0 To 255
    If s(c) <> t(c) Then Debug.Print "Copy attempt failed."
End If

See Also: GlobalAlloc
Category: Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.
GlobalLock Function

Declare Function GlobalLock Lib "kernel32.dll" (ByVal hMem As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GlobalLock locks a moveable block of memory into its current physical position. The block's internal lock count is incremented by one; a memory block is considered locked if its lock counter is greater than zero. Memory blocks cannot be moved or discarded while locked. The function returns a pointer to the beginning of the memory block if successful, or 0 if the function fails.

hMem

The handle to the moveable memory block to lock.

Example:

' Use a block of memory as an intermediary step to copy
' the contents of array s() to array t(). Yes, you could copy them directly,
' but this demonstrates a few different memory functions.
Dim s(0 To 255) As Integer, t(0 To 255) As Integer ' arrays to copy from/to
Dim c As Integer, retval As Long ' counter variable & return value
Dim hMem As Long, pMem As Long ' handle and pointer to memory block

' Initialize the source array s()'s data
For c = 0 To 255
    s(c) = 2 * c ' each element equals double its index
Next c

' Allocate a moveable block of memory (returns a handle) (Integer type = 2 bytes)
hMem = GlobalAlloc (GMEM_MOVEABLE Or GMEM_ZEROINIT, 256 * 2)
' Lock the memory block, returning a pointer to it
pMem = GlobalLock (hMem)
' Copy the entire contents of s() to the memory block
' Note that pMem is ByVal because we want its contents, not a pointer to it
CopyMemory ByVal pMem, s(0), 256 * 2
' Copy the contents of the memory block to t() (we could have just copied s() to t())
CopyMemory t(0), ByVal pMem, 256 * 2
' Unlock the memory block, destroying the pointer and freeing resources
x = GlobalUnlock (hMem)
' Free the memory block (de-allocate it)
x = GlobalFree (hMem)
' Verify that \( t() = s() \), which it should
For \( c = 0 \) To 255
    If \( s(c) \neq t(c) \) Then Debug.Print "Copy attempt failed."
End If

See Also: GlobalUnlock
Category: Memory
GlobalMemoryStatus Function

Declare Sub GlobalMemoryStatus Lib "kernel32.dll" (lpBuffer As MEMORYSTATUS)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported but Obsolete; use GlobalMemoryStatusEx instead.
- **Windows CE**: Unknown.

Description & Usage

The **GlobalMemoryStatus** function retrieves the current status of the computer's memory. It reports both the total memory available and the amount of unused memory. This function only works properly on computers with no more than 4 GB of memory. If the computer has more than 4 GB of memory, the values reported by **GlobalMemoryStatus** are the actual values modulo 4 (for example, a computer with 5 GB total memory would be reported as having only 1 GB of memory). For computers with more than 4 GB of memory, use the **GlobalMemoryStatusEx** function instead.

Return Value

**GlobalMemoryStatusEx** does not return a value.

Visual Basic-Specific Issues

None.

Parameters

* **lpBuffer**
  
  Receives the current status of the computer's memory. The **dwLength** member of the structure does not have to be set before calling the function.

Example

' This code is licensed according to the terms and conditions listed here.

' Display the amounts of total and available physical memory

' on the computer. Also calculate the percentage of used physical memory.
Dim ms As MEMORYSTATUS

' Get the current memory status.
GlobalMemoryStatus ms
' Display total and available physical memory, in KB.
Debug.Print "Total Physical Memory:"; ms.dwTotalPhys \\ 1024; "KB"
Debug.Print "Available Physical Memory:"; ms.dwAvailPhys \\ 1024; "KB"
' Calculate percentage of physical memory in use.
Debug.Print "Used Physical Memory:"; 100 - 100 * ms.dwAvailPhys \\ ms.dwTotalPhys; "%"

See Also

GlobalMemoryStatusEx

Category

Memory

Back to the Function list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/globalmemorystatus.html
GlobalMemoryStatusEx Function

Declare Function GlobalMemoryStatusEx Lib "kernel32.dll" (lpBuffer As MEMORYSTATUSEX) As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Not Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Description & Usage

GlobalMemoryStatusEx retrieves the current status of the computer's memory. It reports both the total memory available and the amount of unused memory.

Return Value

If an error occurred, the function returns 0 (call GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpBuffer

Receives the current status of the computer's memory.

Example
' Display the amounts of total and available physical memory on the computer. Use the Currency data type to allow VB to display the 64-bit values reported by GlobalMemoryStatusEx.
Dim ms As MEMORYSTATUSEX
Dim temp As Currency ' pseudo-64-bit integer buffer
Dim retval As Long ' return value
Dim mult As Long ' conversion multiplier

' Calculate the multiplier necessary to convert bytes in a Currency variable to kilobytes properly displayed:
mult = 10000 / 1024 ' move decimal point, bytes to kilobytes
' Get the current memory status.
retval = GlobalMemoryStatusEx(ms)

' Display total physical memory, in KB.
Debug.Print "Total Physical Memory:";
CopyMemory temp, ms.ullTotalPhys, Len(temp)
Debug.Print temp * mult; "KB"

' Display available physical memory, in KB.
Debug.Print "Available Physical Memory:";
CopyMemory temp, ms.ullAvailPhys, Len(temp)
Debug.Print temp * mult; "KB"

See Also
GlobalMemoryStatus

Category
Memory

Back to the Function list.
Back to the Reference section.
GlobalUnlock Function

Declare Function GlobalUnlock Lib "kernel32.dll" (ByVal hMem As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

GlobalUnlock decrements the lock count of a moveable memory block by one. The memory block is locked if its lock count is greater than zero; it is considered unlocked if its lock count is zero. A moveable memory block cannot be moved or discarded while it is locked. The function returns 1 if, after decrementing the lock count, the memory block is still locked. The function returns 0 if, after decrementing the lock count, either the memory block is now unlocked or an error occurred.

hMem

A handle to the moveable memory block to decrement the lock count of.

Example:

' Use a block of memory as an intermediary step to copy
' the contents of array s() to array t(). Yes, you could copy them directly,
' but this demonstrates a few different memory functions.
Dim s(0 To 255) As Integer, t(0 To 255) As Integer ' arrays to copy from/to
Dim c As Integer, retval As Long ' counter variable & return value
Dim hMem As Long, pMem As Long ' handle and pointer to memory block

' Initialize the source array s()'s data
For c = 0 To 255
    s(c) = 2 * c ' each element equals double its index
Next c

' Allocate a moveable block of memory (returns a handle) (Integer type = 2 bytes)
hMem = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, 256 * 2)
' Lock the memory block, returning a pointer to it
pMem = GlobalLock(hMem)
' Copy the entire contents of s() to the memory block
' Note that pMem is ByVal because we want its contents, not a pointer to it
CopyMemory ByVal pMem, s(0), 256 * 2
' Copy the contents of the memory block to t() (we could have just copied s() to t())
CopyMemory t(0), ByVal pMem, 256 * 2
' Unlock the memory block, destroying the pointer and freeing resources
x = GlobalUnlock(hMem)
' Free the memory block (de-allocate it)
x = GlobalFree(hMem)

' Verify that t() = s(), which it should
For c = 0 To 255
    If s(c) <> t(c) Then Debug.Print "Copy attempt failed."
End If

See Also: GlobalLock
Category: Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.
htonl Function

Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

`htonl` converts a 32-bit value from host byte order to network byte order. Since Winsock uses network byte order for various values, `htonl` ensures that the proper byte order is being used in Winsock calls, regardless of whether the computer normally uses little-endian or big-endian ordering.

Return Value

The function returns the value converted to network byte order.

Visual Basic-Specific Issues

None.

Parameters

`hostlong`

A 32-bit value in host order to convert to network byte order.

Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.
'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type **WSADATA**
  wVersion As Integer
  wHighVersion As Integer
  szDescription As String * 257
  szSystemStatus As String * 129
  iMaxSockets As Long
  iMaxUdpDg As Long
  lpVendorInfo As Long
End Type
Public Declare Function **WSAStartup** Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As **WSADATA**) As Long
Public Declare Function **WSACleanup** Lib "wsock32.dll" () As Long
Public Type **HOSTENT**
  h_name As Long
  h_aliases As Long
  h_addrtype As Integer
  h_length As Integer
  h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function **htonl** Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Function **gethostbyaddr** Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub **CopyMemory** Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Type **INITCOMMONCONTROLSEX_TYPE**
  dwSize As Long
  dwICC As Long
End Type
Public Declare Function **InitCommonControlsEx** Lib "comctl32.dll" (lpInitCtrls As **INITCOMMONCONTROLSEX_TYPE**) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function **CreateWindowEx** Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_ISBLANK = &H469
Public Const IPM_GETADDRESS = &H466

' *** Place the following code in a form window. ***

Private hIPControl As Long  ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the ' upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE  ' identifies the control to register
    Dim retval As Long                        ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
            Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the ' address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long   ' the IP address, in host byte order
    Dim ipAddress_n As Long   ' the IP address, in network byte order
    Dim sockinfo As WSADATA   ' information about the Winsock implementation
    Dim pHostinfo As Long     ' pointer to information about the host computer
    Dim hostinfo As HOSTENT   ' information about the host computer
    Dim domainName As String  ' the primary domain name of the host computer
    Dim retval As Long        ' generic return value
' Verify that an IP address was entered.
retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
If retval <> 0 Then
    Debug.Print "No IP address was entered!"
    Exit Sub
End If

' Get the IP address entered by the user and verify that all
' four fields in the address were entered.
retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
If retval < 4 Then
    Debug.Print "An incomplete IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)
' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also

ntohl

Category

Winsock
htons Function

Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**htons** converts a 16-bit value from host byte order to network byte order. Since Winsock uses network byte order for various values, **htons** ensures that the proper byte order is being used in Winsock calls, regardless of whether the computer normally uses little-endian or big-endian ordering.

Return Value

The function returns the value converted to network byte order.

Visual Basic-Specific Issues

None.

Parameters

**hostshort**

A 16-bit network address in host order to convert to network byte order.

Example

Download the main page of this web site ([http://www.vbapi.com](http://www.vbapi.com)). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.
Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

'This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer
Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long
Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type
Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long
Declared Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long
Public Const FIONBIO = &H8004667E
Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, _
Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, _
    ByVal flags As Long) As Long

Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source _
    As Any, ByVal Length As Long)

Public Const SOCKET_ERROR = -1

' Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA  ' info about Winsock
    Dim sock As Long          ' the socket descriptor
    Dim pHostinfo As Long     ' pointer to info about the host computer
    Dim hostinfo As hostent   ' info about the host computer
    Dim pIPAddress As Long    ' pointer to host's IP address
    Dim ipAddress As Long     ' host's IP address
    Dim sockinfo As sockaddr  ' settings for the socket
    Dim buffer As String      ' buffer for sending and receiving data
    Dim reply As String       ' accumulates server's reply
    Dim retval As Long        ' generic return value

    ' Begin a Winsock session.
    retval = WSAStartup(MAKEWORD(2, 2), wsockinfo)
    If retval <> 0 Then
        Debug.Print "Unable to initialize Winsock! --"; retval
        Exit Sub
    End If

    ' Get information about the server to connect to.
    pHostinfo = gethostbyname("www.vbapi.com")
    If pHostinfo = 0 Then
        Debug.Print "Unable to resolve host!"
        GoTo Cleanup
    End If

    ' Copy information about the server into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print "Couldn't get IP address of www.vbapi.com!"
        GoTo Cleanup
    End If

    ' Get the server's IP address out of the structure.
    CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4

' Create a socket.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If

' Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
    ' Use Internet Protocol (IP)
    .sin_family = AF_INET
    ' Connect to port 80.
    .sin_port = htons(80)
    ' Connect to this IP address.
    .sin_addr = ipAddress
    ' Padding characters.
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If

' Send an HTTP/GET request for the / document.
buffer = "GET / HTTP/1.1" & vbCrLf & _
    "Host: www.vbapi.com" & vbCrLf & _
    "User-Agent: HTTP-Test-Program" & vbCrLf _
    retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request.  Waiting for reply..."

' Make the socket non-blocking, so calls to recv don't halt the program
waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)

' Read the response from the other system. A more sophisticated program
' would watch to see if the connection ever times out (i.e., if the
' connection is
' lost). For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If
    DoEvents
Loop Until retval = 0
' Print the response from the server.
Debug.Print "Document Retrieved:"
Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()
End Sub

See Also

htonl

Category

Winsock

Go back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/h/htons.html
inet_addr Function

Declare Function inet_addr Lib "wsock32.dll" (ByVal cp As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

inet_addr converts a string identifying an IP address into an actual IP address in network byte order. The string must be in "dotted quad" format (for example, "a.b.c.d").

Return Value

If successful, the function returns the IP address, packed into a single 32-bit integer, in network byte order. If an error occurred, the function returns &HFFFFFFFF.

Visual Basic-Specific Issues

None.

Parameters

`cp`

The IP address string to convert. This must be in "dotted quad" format.

Example

Prompt the user for an IP address by using a regular text box. Take the IP address and determine the domain name, if any, associated with that address.

To use this example, place a command button named cmdGetDomain on a form window. Also place a text box named txtIPAddress on the form window.
' Declarations and such needed for the example:
  ' (Copy them to the (declarations) section of a module.)
Public Type **WSADATA**
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function **WSAStartup** Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function **WSACleanup** Lib "wsock32.dll" () As Long
Public Type **HOSTENT**
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function **inet_addr** Lib "wsock32.dll" (ByVal cp As String) As Long
Public Declare Function **gethostbyaddr** Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub **CopyMemory** Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function **lstrlen** Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function **lstrcpy** Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

' *** Place the following code in a form window. ***

' Look up the primary domain name of the host computer identified by the address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_n As Long   ' the IP address, in network byte order
    Dim sockinfo As WSADATA   ' information about the Winsock implementation
    Dim pHostinfo As Long     ' pointer to information about the host computer
    Dim hostinfo As HOSTENT   ' information about the host computer
    Dim domainName As String  ' the primary domain name of the host computer
    Dim retval As Long        ' generic return value

    ' Get the IP address entered by the user.

ipAddress_n = inet_addr(txtIPAddress.Text)
If ipAddress_n = &HFFFFFFFF Then
    Debug.Print "An incorrect IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also
inet_ntoa

Category
Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/inet_addr.html
inet_ntoa Function

Declare Function inet_ntoa Lib "wsock32.dll" (ByVal inaddr As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

inet_ntoa converts an IP address packed in a 32-bit integer into a human-readable string. This string is in the standard "a.b.c.d" format. This function returns a pointer to the resulting string. However, the pointer is only guaranteed to be valid until the next call to a Winsock function, so your program should make a copy of it immediately.

Return Value

If successful, the function returns a pointer to the string that contains the human-readable IP address. If an error occurred, the function returns zero.

Visual Basic-Specific Issues

To copy the returned string into a String variable, use the lstrlen and lstrcpy API functions. See the example below for a demonstration.

Parameters

inaddr

The IP address to convert to a human-readable string. The IP address is packed into a single 32-bit integer and must be in network byte order.

Example

Print the IP address associated with a domain name specified by the user. Winsock is briefly used to resolve the domain name and format a printable IP address string. The user enters the domain name to resolve in text box txtDomain, and the domain...
name is resolved when the user clicks button cmdGetIP.

To run this example, place a text box named txtDomain and a command button named cmdGetIP on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function inet_ntoa Lib "wsock32.dll" (ByVal inaddr As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

' Define a relevant API macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdGetIP_Click()
Dim sockinfo As WSADATA ' information about Winsock
Dim hostinfo As HOSTENT ' information about an Internet host
Dim pHostinfo As Long ' pointer to a HOSTENT structure
Dim ipAddress As Long ' pointer to an IP address dword
Dim pIPAddress As Long ' an IP address, packed into a dword
Dim pIPString As Long ' pointer to an IP address formatted as a string
Dim ipString As String ' holds a human-readable IP address string
Dim retval As Long ' generic return value

' Open up a Winsock session, using version 2.2.
retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(txtDomain.Text)
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve domain name."
Else
    ' Copy the data into a HOSTENT structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print "A non-IP address was returned."
    Else
        ' Copy the pointer to the first (and probably only) IP
        address in the structure.
        CopyMemory ipAddress, ByVal hostinfo.h_addr_list, 4
        ' Copy the actual IP address.
        CopyMemory pIPAddress, ByVal ipAddress, 4
        ' Convert the IP address into a human-readable string.
        pIPString = inet_ntoa(ipAddress)
        ' Copy the result into a string variable.
        ipString = Space(lstrlen(pIPString))
        retval = lstrcpy(ipString, pIPString)
        ' Print the result: a human-readable IP address.
        Debug.Print ipString
    End If
End If

' Close the Winsock session.
retval = WSACleanup()
End Sub

See Also

inet_addr

Category

Winsock

Back to the Function list.
Back to the Reference section.
InflateRect Function

Declare Function InflateRect Lib "user32.dll" (lpRect As RECT, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

InflateRect increases or decreases the size of a rectangle. The values to inflate the rectangle by are added to both sides of it, so in reality the width and/or height of the rectangle increases by double what you pass to the function. For example, if you pass 20 as x, the left and right sides will both be extended by 20, so the total width will be 40 more. Positive values increase the size, while negative values decrease it. The function returns 0 if an error occurred, or 1 if successful.

lpRect
The rectangle to change the size of.

x
The value to expand the left and right sides by. Positive values increase the width; negative values decrease it.

y
The value to expand the top and bottom by. Positive values increase the height; negative values decrease it.

Example:

' Expand the width of window Form1 by 100 and shrink its height by 50 using its rectangle.
Dim winrect As RECT  ' receives the rectangle of the window
Dim retval As Long  ' return value

retval = GetWindowRect(Form1.hWnd, winrect)  ' get Form1's rectangle
retval = InflateRect(winrect, 50, -25)  ' these values added to each side to inflate it

' Now change the window on screen to match its new rectangle
retval = SetWindowPos(Form1.hWnd, 0, winrect.Left, winrect.Top, winrect.Right, winrect.Bottom, 0)

See Also: OffsetRect
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
InitCommonControlsEx Function

Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

InitCommonControlsEx registers the window classes used to create various common controls. This function must be called before using CreateWindowEx to create a common control.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

lpInitCtrls

Specifies which common control classes to register.

Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/i/initcommoncontrolsex.html).

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

```vba
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type

Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long

Public Declare Function WSACleanup Lib "wsock32.dll" () As Long

Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type

Public Const AF_INET = 2

Public Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long

Public Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)

Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long

Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type

Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long

Public Const ICC_INTERNET_CLASSES = &H800

Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long
```
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_ISBLANK = &H469
Public Const IPM_GETADDRESS = &H466

' *** Place the following code in a form window. ***

Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
    Dim retval As Long ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long ' the IP address, in host byte order
    Dim ipAddress_n As Long ' the IP address, in network byte order
    Dim sockinfo As WSADATA ' information about the Winsock implementation
    Dim pHostinfo As Long ' pointer to information about the host computer
    Dim hostinfo As HOSTENT ' information about the host computer
    Dim domainName As String ' the primary domain name of the host computer
    Dim retval As Long ' generic return value
' Verify that an IP address was entered.
    retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
    If retval <> 0 Then
        Debug.Print "No IP address was entered!"
        Exit Sub
    End If

' Get the IP address entered by the user and verify that all
' four fields in the address were entered.
    retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
    If retval < 4 Then
        Debug.Print "An incomplete IP address was entered!"
        Exit Sub
    End If

' Open up a Winsock v2.2 session.
    retval = WSAStartup(&H202, sockinfo)
    If retval <> 0 Then
        Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
        Exit Sub
    End If

' Convert the IP address into network byte order.
    ipAddress_n = htonl(ipAddress_h)
' Get information about the host computer.
    pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
    If pHostInfo = 0 Then
        Debug.Print "Could not find a host with the specified IP address."
    Else
        ' Copy the data into the structure.
        CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
        ' Copy the host domain name into a string.
        domainName = Space(lstrlen(hostinfo.h_name))
        retval = lstrcpy(domainName, hostinfo.h_name)
        Debug.Print "Domain name is: "; domainName
    End If

' End the Winsock session.
    retval = WSACleanup()
InsertMenuItem Function

Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

InsertMenuItem adds a menu item to a menu that already exists. The new item can be inserted at any point in the menu. If the menu is a regular menu, then the window it belongs to will receive a WM_COMMAND message whenever the menu item is selected. If the menu is a window's system menu, then the window receives the WM_SYSCOMMAND message instead.

Also, when adding an item to a window's system menu, be sure that the new item's unique menu item identifier is less than &HF000. All identifiers greater than that are reserved for items that the system places there (such as Restore, Minimize, etc.).

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

When using InsertMenuItem to add something to a menu, keep in mind that there is no way for Visual Basic to create an event handler to run when the item is selected. Instead, it is necessary to create a WindowProc window procedure to manually process the WM_COMMAND or WM_SYSCOMMAND messages that the menu item may send. See the example below for a demonstration of how to accomplish this.

Parameters

hMenu

A handle to the menu to add a new item to.
InsertMenuItem Function

Identifies the existing menu item that appears immediately before the point where you want to add the new item. This could be either a position or a menu item identifier, depending on `fByPosition`.

If this is a non-zero value, `uItem` indicates the existing item by using its zero-based position. (For example, the first item in the menu has a position of 0.) If this is zero, then `uItem` is the unique menu item identifier of the existing item.

Describes the menu item to be added.

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
' There's quite a few declarations for this example, but it's worth it!
Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1

Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Public Const GWL_WNDPROC = -4

Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116

' Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.
' Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

' *** Place the following code in a module. ***

Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public ontop As Boolean  ' identifies if Form1 is always on top or not

' The following function acts as Form1's window procedure to process messages.
Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Dim hSysMenu As Long     ' handle to Form1's system menu
Dim mii As MENUITEMINFO  ' menu item information for Always On Top
Dim retval As Long       ' return value

Select Case uMsg
Case WM_INITMENU
  ' Before displaying the system menu, make sure that the Always On Top option is properly checked.
  hSysMenu = GetSystemMenu(hwnd, 0)
  With mii
    .cbSize = Len(mii)
    .fMask = MIIM_STATE
    .fState = MFS_ENABLED Or If(ontop, MFS_CHECKED, 0)
  End With
  retval = SetMenuInfo(hSysMenu, 1, 0, mii)
  WindowProc = 0
Case WM_SYSCOMMAND
  ' If Always On Top (ID = 1) was selected, change the on top/not on
  ' setting of Form1 to match.
  If wParam = 1 Then
    ' Reverse the setting and make it the current one.
    ontop = Not ontop
    retval = SetWindowPos(hwnd, If(ontop, HWND_TOPMOST, 
      HWND_NOTOPMOST), _
      0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
    WindowProc = 0
  Else
    ' Some other item was selected. Let the previous window
    ' procedure process it.
    WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam,
      lParam)
  End If
End Select
End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the
' new window procedure.
Private Sub Form_Load()
  Dim hSysMenu As Long     ' handle to the system menu
  Dim count As Long        ' the number of items initially on the menu
  Dim mii As MENUITEMINFO  ' describes a menu item to add
  Dim retval As Long       ' return value

  ' Get a handle to the system menu.
  hSysMenu = GetSystemMenu(Form1.hWnd, 0)
  ' See how many items are currently in it.
  count = GetMenuItemCount(hSysMenu)

  ' Add a separator bar and then Always On Top to the system menu.
  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With
  ' Add the separator to the end of the system menu.

  ' Add Always On Top to the system menu.
  mii.fMask = MIIM_ID
  mii.wID = 1
  mii.dwTypeData = "Always On Top"
  mii.dwFontSize = 0
  mii.fMask = MIIM_ID Or MIIM_TYPE
  mii.cchSzText = Len("Always On Top")
  mii.cchMenuPrefInt = 0

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With
  ' Add the separator to the end of the system menu.

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
  End With

  With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
retval = InsertMenuItem(hSysMenu, count, 1, mii)

' Likewise, add the Always On Top command.
With mii
    .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        ' This is a regular text item.
    .fType = MFT_STRING
        ' The option is enabled.
    .fState = MFS_ENABLED
        ' It has an ID of 1 (this identifies it in the window procedure).
    .wID = 1
        ' The text to place in the menu item.
    .dwTypeData = "&Always On Top"
    .cch = Len(.dwTypeData)
End With
' Add this to the bottom of the system menu.
retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)

' Set the custom window procedure to process Form1's messages.
ontop = False
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' Before unloading, restore the default system menu and remove the
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value

    ' Replace the previous window procedure to prevent crashing.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
    ' Remove the modifications made to the system menu.
    retval = GetSystemMenu(Form1.hWnd, 1)
End Sub

Category

Menus

Back to the Function list.
Back to the Reference section.
IntersectRect Function

Declare Function IntersectRect Lib "user32.dll" (lpDestRect As RECT, lpSrc1Rect As RECT, lpSrc2Rect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

IntersectRect creates a rectangle based on the intersecting part of two other rectangles. The rectangular region where the two source rectangles overlap is the intersection rectangle. If one or both of the source rectangles are empty or there is no intersection, the function returns 0 and the lpDestRect rectangle is set to (0,0)-(0,0). If the source rectangles do overlap, the intersection is put into lpDestRect and the function returns 1.

lpDestRect
The rectangle to be set as the intersection of the two source rectangles.

lpSrc1Rect
The first source rectangle.

lpSrc2Rect
The second source rectangle.

Example:

' Determine if windows Form1 and Form2 are overlapping on the screen. If they don't the intersection rectangle will be empty.
Dim intrect As RECT ' receives the intersection rectangle
Dim window1 As RECT, window2 As RECT ' receive rectangles of Form1 and Form2
Dim result As Long ' will be set to 0 if no intersection, 1 if there is intersection
Dim retval As Long ' return value for other functions

retval = GetWindowRect(Form1.hWnd, window1) ' get Form1's rectangle
retval = GetWindowRect(Form2.hWnd, window2) ' get Form2's rectangle
result = IntersectRect(intrect, window1, window2) ' determine the intersection rectangle
If result = 0 ' in this case, intrect will also be empty
    Debug.Print "Windows Form1 and Form2 are not overlapping on the screen."
Else
    Debug.Print "Windows Form1 and Form2 are overlapping on the screen."
End If

See Also: SubtractRect, UnionRect

Category: Rectangles
InvertRect Function

Declare Function InvertRect Lib "user32.dll" (ByVal hdc As Long, ByVal lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

InvertRect inverts the image inside a rectangular area on a device. The inverted colors are calculated by taking the binary NOT of each pixel's RGB color value inside the rectangle. The function returns a non-zero value if successful, or 0 if an error occurred.

*hdch*  
A device context to the device to invert the colors in a given rectangle on.

*lpRect*  
The rectangle to invert.

Example:

' Invert the colors in the rectangle (20,30)-(150,100) on window Form1.  
Dim r As RECT  ' rectangle to invert  
Dim retval As Long  ' return value  
retval = SetRect(r, 20, 30, 150, 100)  ' set r to be (20,30)-(150,100)  
retval = InvertRect(Form1.hDC, r)  ' invert the pixels within the rectangle

See Also: InvertRgn
Category: Filled Shapes

Go back to the alphabetical Function listing.  
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.  
Go back to the Windows API Guide home page.  
E-mail: vbapi@vbapi.com Send Encrypted E-Mail  
This page is at http://www.vbapi.com/ref/i/invertrect.html
InvertRgn Function

Declare Function InvertRgn Lib "gdi32.dll" (ByVal hdc As Long, ByVal hRgn As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

InvertRgn inverts the colors of all the pixels inside a region on a given device. The inverse of the pixels is calculated by doing a binary NOT operation on the RGB color value of each pixel. The function returns 0 if an error occurred, or a non-zero value if successful.

hdc
A device context to the device to invert the pixels within a region of.

hRgn
A handle to the region on the device to invert.

Example:

' Invert the pixels within an elliptical region on window Form1. The elliptical region has a bounding rectangle of (20,30)-(150,110).
Dim hrgn As Long  ' handle to the region to invert
Dim retval As Long  ' return value

' Create the elliptical region to invert and get a handle to it.
    hrgn = CreateEllipticRgn(20,30,150,110)
' Invert that region in window Form1.
    retval = InvertRgn(Form1.hDC, hrgn)
' Delete the region to free up resources.
    retval = DeleteObject(hrgn)

See Also: InvertRect
Category: Regions
ioctlsocket Function

Declare Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

`ioctlsocket` manipulates the i/o mode of a socket. It gets or sets parameters that control how the socket performs input and output. Naturally, the socket must have been created by a previous call to `socket`.

Return Value

If successful, the function returns zero. If an error occurred, the function returns SOCKET_ERROR (use `WSAGetLastError` to get the error code).

Visual Basic-Specific Issues

When setting `cmd` to FIONBIO, do not use the ByVal keyword in front of the parameter passed as `argp`.

Parameters

- **s**
  The descriptor of the socket to manipulate.

- **cmd**
  One of the following flags specifying the action to take:
  - **FIONBIO**
    Set the blocking mode of the socket. If `argp` is zero, the socket is set to block (halt the program if no data is yet available to read from the socket), which is the default value. If this is a non-zero value, the socket is set to nonblocking mode, where functions reading the socket return immediately instead of waiting for input to arrive.
  - **FIONREAD**
    Determine the amount of data sitting in the socket's input buffer. The variable passed as `argp` receives the number of bytes sitting in the buffer waiting to be read. (For datagram-based sockets, this value is the size of the
SIOCATMARK
Determine if there is any out-of-band data waiting to be read from the socket. If so, the variable passed as argp receives zero, and the next attempt to read from the socket will read from the out-of-band data. If there is no unread out-of-band data, argp receives a non-zero value.

argp
Usage depends on the flag used for cmd.

Constant Definitions

Const FIONBIO = &H8004667E
Const FIONREAD = &H4004667F
Const SIOCATMARK = &H40047307
Const SOCKET_ERROR = -1

Example

Download the main page of this web site (http://www.vbapi.com). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent

h_name As Long
h_aliases As Long
h_addrtype As Integer
h_length As Integer
h_addr_list As Long

End Type

Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer

Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long

Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type

Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long

Declare Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long

Public Const FIONBIO = &H8004667E

Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long

Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

Public Const SOCKET_ERROR = -1

' Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA ' info about Winsock
    Dim sock As Long          ' the socket descriptor
    Dim pHostinfo As Long     ' pointer to info about the host computer
    Dim hostinfo As hostent   ' info about the host computer
    Dim pIPAddress As Long    ' host's IP address
    Dim ipAddress As Long     ' host's IP address
    Dim sockinfo As sockaddr  ' settings for the socket

Dim buffer As String      ' buffer for sending and receiving data
Dim reply As String       ' accumulates server's reply
Dim retval As Long        ' generic return value

' Begin a Winsock session.
retval = WSAStartup(MAKEWORD(2, 2), wsockinfo)
If retval <> 0 Then
    Debug.Print "Unable to initialize Winsock! --"; retval
    Exit Sub
End If

' Get information about the server to connect to.
pHostinfo = gethostbyname("www.vbapi.com")
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve host!"
    GoTo Cleanup
End If

' Copy information about the server into the structure.
CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
If hostinfo.h_addrtype <> AF_INET Then
    Debug.Print "Couldn't get IP address of www.vbapi.com!"
    GoTo Cleanup
End If

' Get the server's IP address out of the structure.
CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4

' Create a socket.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If

' Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
    .sin_family = AF_INET
    .sin_port = htons(80)
    .sin_addr = ipAddress
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If
' Send an HTTP/GET request for the / document.
buffer = "GET / HTTP/1.1" & vbCrLf & 
"Host: www.vbapi.com" & vbCrLf & 
"User-Agent: HTTP-Test-Program" & vbCrLf & vbCrLf
retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request. Waiting for reply..."

' Make the socket non-blocking, so calls to recv don't halt the program waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)

' Read the response from the other system. A more sophisticated program would watch to see if the connection ever times out (i.e., if the connection is lost). For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If
    ' Process background events so the program doesn't appear to freeze.
    DoEvents
Loop Until retval = 0

' Print the response from the server.
Debug.Print "Document Retrieved:"
Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()

End Sub

Category

Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/ioctlsocket.html
IsChild Function

Declare Function IsChild Lib "user32.dll" (ByVal hWndParent As Long, ByVal hWnd As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

IsChild determines if a parent-child relationship exists between two windows. The possible child window must be a direct descendant of the possible parent window. For example, if the possible child window is a child of a child of the possible parent window, the parent-child relationship does not exist.

Return Value

If a parent-child relationship does not exist between the two windows, the function returns 0. If the relationship does exist, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- `hWndParent`  
  A [handle](https://www.vbapi.com) to the candidate parent window.

- `hWnd`  
  A handle to the candidate child window.

Example
'Demonstrate a parent-child relationship and a non-parent-child relationship. Command1, a command button on window Form1, is a child of Form1. However, fellow window Form2 is not a child.

Dim result As Long ' result of the function

' Verify that Command1 is a child of Form1.
result = IsChild(Form1.hWnd, Command1.hWnd) ' see if Form1 is Command1's parent
If result = 0 Then
    Debug.Print "Form1 is not Command1's parent window." ' won't happen
Else
    Debug.Print "Form1 is Command1's parent window." ' will happen
End If

' Verify that Form2 is not a child of Form1.
result = IsChild(Form1.hWnd, Form2.hWnd) ' see if Form1 is Form2's parent
If result = 0 Then
    Debug.Print "Form1 is not Form2's parent window." ' will happen
Else
    Debug.Print "Form1 is Form2's parent window." ' won't happen

See Also
IsWindow, SetParent

Category
Windows

Back to the index.

Last Modified: August 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/ischild.html
IsIconic Function

Declare Function IsIconic Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

IsIconic finds if a given window is minimized or not. The function returns 0 if the window is not minimized (it could be either restored or maximized); it returns a non-zero value if the window is minimized.

hwnd

A handle to the window to find whether it is minimized or not.

Example:

' Determine if the window Form1 is maximized, minimized, or restored.
Dim minflag As Long, maxflag As Long  ' receive minimized or maximized status

minflag = IsIconic(Form1 hWnd)  ' is Form1 minimized?
maxflag = IsZoomed(Form1 hWnd)  ' is Form1 maximized?

If minflag <> 0 Then
    Debug.Print "Form1 is minimized."
ElseIf maxflag <> 0 Then
    Debug.Print "Form1 is maximized."
Else
    Debug.Print "Form1 is restored."
End If

See Also: IsZoomed, ShowWindow

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
Windows API Guide: IsIconic Function

This page is at http://www.vbapi.com/ref/i/isiconic.html
IsRectEmpty Function

Declare Function IsRectEmpty Lib "user32.dll" (lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

IsRectEmpty checks to see if a rectangle is empty. A rectangle is considered empty if its right edge is not to the right of its left edge and/or its bottom edge is not below its top edge. For example, a rectangle (50,50)-(25,100) is empty because the right edge is to the left of the left edge. The function returns 1 if the rectangle is empty and 0 if it is not.

lpRect
  The rectangle to check.

Example:

' Determine if windows Form1 and Form2 are overlapping on the ' screen. If they don't the intersection rectangle will be empty.
Dim intrect As RECT ' receives the intersection rectangle
Dim window1 As RECT, window2 As RECT ' receive rectangles of Form1 and Form2
Dim isempty As Long ' will be set to 0 if intersection isn't empty, 1 if it is
Dim retval As Long ' return value for other functions
retval = GetWindowRect(Form1.hWnd, window1) ' get Form1's rectangle
retval = GetWindowRect(Form2.hWnd, window2) ' get Form2's rectangle
retval = IntersectRect(intrect, window1, window2) ' determine the intersection rectangle
isempty = IsRectEmpty(intrect) ' determine if it is empty -- it will be if there's no intersection
If isempty = 0 ' in this case, intrect will also be empty
    Debug.Print "Windows Form1 and Form2 are not overlapping on the screen."
Else
    Debug.Print "Windows Form1 and Form2 are overlapping on the screen."
End If

See Also: SetRectEmpty
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
IsWindow Function

Declare Function IsWindow Lib "user32.dll" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

IsWindow determines if a given handle refers to a window or not. The function verifies that the handle in fact refers to a window, instead of one of the many other objects which handles can represent.

Return Value

If the handle does not refer to a window, the function returns 0. If the handle does refer to a window, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

hWnd

The handle to check if it refers to a window or not.

Example
This code is licensed according to the terms and conditions listed here.

Demonstrate handles refering to a window and not refering to a window.
Dim hWindow As Long  ' handle to a window (Form1, to be exact)
Dim hPen As Long  ' handle to a pen (the solid black stock pen)
Dim result As Long  ' result of the test

' Initialize the two handles.
hWindow = Form1.hWnd  ' hWindow now equals window Form1's handle
hPen = GetStockObject(BLACK_PEN)  ' hPen refers to a pen
' Verify that hWindow refers to a window.
result = IsWindow(hWindow)
If result = 0 Then
    Debug.Print "hWindow does not refer to a window."  ' won't happen
Else
    Debug.Print "hWindow refers to a window."  ' will happen
End If
' Verify that hPen does not refer to a window.
result = IsWindow(hPen)
If result = 0 Then
    Debug.Print "hPen does not refer to a window."  ' will happen
Else
    Debug.Print "hPen refers to a window."  ' won't happen
End If

See Also
IsChild

Category
Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000 Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
Windows API Guide: IsWindow Function

This page is at http://www.vbapi.com/ref/i/iswindow.html
IsWindowEnabled Function

Declare Function IsWindowEnabled Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

IsWindowEnabled determines if a window is currently enabled or disabled. If a window is disabled, it cannot receive the focus and will ignore any attempted input. Many types of windows, such as buttons and other controls, will appear grayed when disabled. The function returns 0 if the window is disabled, or a non-zero value if the window is enabled.

hwnd

A handle to the window to determine if it is enabled or disabled.

Example:

' Reverse the enabled status of window Command1. If the window is disabled, enable it; if it is enabled, disable it.
Dim wasenabled As Long  ' receives enabled/disabled status of Command1
Dim retval As Long  ' return value

' Determine if the window Command1 is currently enabled or not.
wasenabled = IsWindowEnabled(Command1.hWnd)
If wasenabled = 0 Then  ' if not enabled, enable it
    retval = EnableWindow(Command1.hWnd, 1)
Else  ' if enabled, disable it
    retval = EnableWindow(Command1.hWnd, 0)
End If

See Also: EnableWindow
Category: Windows
IsZoomed Function

Declare Function IsZoomed Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

IsZoomed finds if a given window is maximized or not. The function returns 0 if the window is not maximized (it could be either restored or minimized); it returns a non-zero value if the window is maximized.

hwnd
A handle to the window to find whether it is maximized or not.

Example:

' Determine if the window Form1 is maximized, minimized, or restored.
Dim minflag As Long, maxflag As Long  ' receive minimized or maximized status
minflag = IsIconic(Form1.hWnd)  ' is Form1 minimized?
maxflag = IsZoomed(Form1.hWnd)  ' is Form1 maximized?
If minflag <> 0 Then
    Debug.Print "Form1 is minimized."
ElseIf maxflag <> 0 Then
    Debug.Print "Form1 is maximized."
Else
    Debug.Print "Form1 is restored."
End If

See Also: IsIconic, ShowWindow
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
Windows API Guide: IsZoomed Function

This page is at http://www.vbapi.com/ref/i/iszoomed.html
joyGetDevCaps Function

Declare Function joyGetDevCaps Lib "winmm.dll" Alias "joyGetDevCapsA" (ByVal id As Long, lpCaps As JOYCAPS, ByVal uSize As Long) As Long

Platforms: Win 95/98

joyGetDevCaps reads various information about a joystick. This information is put into the variable passed as lpCaps. This function does not, however, give you the current position of the joystick. The function returns 0 if the joystick is connected and working and a non-zero error code if it isn't.

id
The ID number of the joystick to read, starting with 0.

lpCaps
Variable that receives the information about the joystick.

uSize
The length in bytes of lpCaps.

Example:

' Display the name of the joystick driver for Joystick #1
' Note that the ID of Joystick #1 is 0.
Dim joyinfo As JOYCAPS  ' receives joystick information
Dim joydriver As String  ' will be set to the joystick's driver name
Dim retval As Long  ' return value

retval = joyGetDevCaps(0, joyinfo, Len(joyinfo))  ' read joystick information
If retval = 0 Then  ' there is a functioning Joystick #1
    joydriver = Left(joyinfo.szPname, InStr(joyinfo.szPname, vbNullChar) - 1)  ' extract data from the fixed-length string
    Debug.Print "The joystick driver is: "; joydriver
Else
    Debug.Print "There is no joystick connected to Joystick Port #1."
End If

Category: Joysticks

Go back to the alphabetical Function listing.
Go back to the Reference section index.
joyGetNumDevs Function

Declare Function joyGetNumDevs Lib "winmm.dll" () As Long

Platforms: Win 95/98

joyGetNumDevs returns the number of joysticks that are configured under Windows's Control Panel. This doesn't necessarily mean that all of them are connected and in working order, but there could be. The best way to see if a joystick is working is to call joyGetDevCaps and check for a return value of 0.

Example:

' Determine the number of configured joysticks and the number of ' connected joysticks. To see if a joystick is connected, try to read information from it and see if ' the attempt is successful or not.
Dim joyinfo As JOYCAPS ' needed for the function call to see if a joystick works
Dim numjoys As Long ' receives number of configured joysticks
Dim numexist As Long ' number of existing joysticks hooked up to the computer
Dim c As Integer ' counter variable
Dim retval As Long ' return value for other functions
numjoys = joyGetNumDevs() ' determine the number of configured joysticks
Debug.Print "There are"; numjoys; "joysticks configured under Windows."
numexist = 0 ' initialize the number of existing joysticks
For c = 0 To numjoys - 1 ' check each joystick (remember Joystick #1's ID = 0, etc.)
    retval = joyGetDevCaps(c, joyinfo, Len(joyinfo)) ' try to read information
    If retval = 0 Then numexist = numexist + 1 ' increment counter if the joystick is connected
Next c
Debug.Print "There are"; numexist; "joysticks currently connected to the computer."

Category: Joysticks

Go back to the alphabetical Function listing.
Go back to the Reference section index.
joyGetPos Function

Declare Function joyGetPos Lib "winmm.dll" (ByVal uJoyID As Long, pji As JOYINFO) As Long

Platforms: Win 95/98

joyGetPos reads the current position and status of a joystick. This information is put into the variable passed as pji. The function returns 0 if the joystick is connected and working, or a non-zero error code if it is not.

uJoyID
   The ID number of the joystick to read, starting at 0.

pji
   Variable that receives the joystick's current position and status.

Example:

' Display the x, y, and z coordinates of Joystick #1, as well
' as the status of buttons 1-4 (which are the only ones this function can read).
Dim joypos As JOYINFO  ' receives current joystick status
Dim retval As Long  ' return value
retval = joyGetPos(0, joypos)  ' get the joystick status (remember Joystick #1's ID = 0)
Debug.Print "X Coordinate:"; joypos.wXpos
Debug.Print "Y Coordinate:"; joypos.wYpos
Debug.Print "Z Coordinate:"; joypos.wZpos
If (joypos.wButtons And JOY_BUTTON1) = JOY_BUTTON1 Then Debug.Print "Button 1 is depressed."
If (joypos.wButtons And JOY_BUTTON2) = JOY_BUTTON2 Then Debug.Print "Button 2 is depressed."
If (joypos.wButtons And JOY_BUTTON3) = JOY_BUTTON3 Then Debug.Print "Button 3 is depressed."
If (joypos.wButtons And JOY_BUTTON4) = JOY_BUTTON4 Then Debug.Print "Button 4 is depressed."

Category: Joysticks

Go back to the alphabetical Function listing.
Go back to the Reference section index.
keybd_event Function

Declare Sub keybd_event Lib "user32.dll" (ByVal bVk As Byte, ByVal bScan As Byte, ByVal dwFlags As Long, ByVal dwExtraInfo As Long)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported but Obsolete; use SendInput instead.
- **Windows NT**: Requires Windows NT 3.1 or later but Obsolete with Windows NT 4.0 with Service Pack 3 (SP3) or later; use SendInput instead.
- **Windows 2000**: Supported but Obsolete; use SendInput instead.
- **Windows CE**: Requires Windows CE 1.0 or later but Obsolete with Windows CE 2.0 or later; use SendInput instead.

Description & Usage

keybd_event simulates keyboard input by placing a keyboard input event into the input stream. The function can simulate a single press or release of a single key. This function should only be used when a key's state changes. For example, do not tell the function to simulate pressing the Z key if the Z key is already pressed.

Return Value

keybd_event does not return a value.

Visual Basic-Specific Issues

None.

Parameters

- **bVk**
  
  The virtual-key code of the key to simulate pressing or releasing.
Windows API Guide: keybd_event Function

bScan
Reserved -- set to 0.

dwFlags
A combination of the following flags specifying what kind of keyboard input to synthesize:

KEYEVENTF_EXTENDEDKEY
Prefix the scan code with a prefix byte having the value &HE0.

KEYEVENTF_KEYUP
The key specified in bVk is being released. If this flag is not specified, the key is being pressed.

dwExtraInfo
An additional 32-bit value associated with the keyboard event.

Constant Definitions

Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2

Example

' This code is licensed according to the terms and conditions listed here.

' Simulate the user pressing Alt+Space followed by N. This key combination will minimize the active window.

' Hold the Alt key while typing Space.
keybd_event VK_MENU, 0, 0, 0  ' press Alt
keybd_event VK_SPACE, 0, 0, 0  ' press Space
keybd_event VK_SPACE, 0, KEYEVENTF_KEYUP, 0  ' release Space
keybd_event VK_MENU, 0, KEYEVENTF_KEYUP, 0  ' release Alt

' Type the N key.
keybd_event VK_N, 0, 0, 0  ' press N
keybd_event VK_N, 0, KEYEVENTF_KEYUP, 0  ' release N

See Also

mouse_event, SendInput

Category

Keyboard
KillTimer Function

Declare Function KillTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

KillTimer deactivates and destroys the specified timer. However, it does not remove any existing WM_TIMER messages which may still be sitting in a window's message queue, if the timer was configured to do so. Your program should destroy any timer created by SetTimer once it is no longer needed.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to the window that owns the timer. This must be the same value originally passed to SetTimer.

uIDEvent
If the timer was owned by a window, this is the program-defined identifier of the timer. This index was specified in the call to SetTimer. If the timer is not owned by any window, this is the value returned by the call to SetTimer.

Example

Display the current time in text box control Text1. The time is updated twice every second, and the time is formatted according
to the current locale's settings. To use this example, place a text edit box named Text1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type
Public Declare Function SetTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc As Long) As Long
Public Declare Function KillTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long) As Long
Public Declare Function GetTimeFormat Lib "kernel32.dll" Alias "GetTimeFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpTime As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpTimeStr As String, ByVal cchTime As Long) As Long

' *** Place the following code inside a module. ***

' The following function will execute twice every second. It retrieves
' the current time and displays it according to the current locale's formatting
' preferences.
Public Sub TimerProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal idEvent As Long, ByVal dwTime As Long) 
    Dim systime As SYSTEMTIME    ' the current time
    Dim timestr As String * 260  ' receives the formatted string
    Dim slength As Long          ' length of formatted string returned
    GetLocalTime systime
    slength = GetTimeFormat(0, 0, systime, Clng(0), timestr, Len(timestr))
    Form1.Text1.Text = Left(timestr, slength)
End Function

' *** Place the following code inside Form1. ***

' Create the timer when the form opens and destroy it when the form closes.
' The timer is given an ID of 1, so the return values don't need to be saved.
Private Sub Form1_Load()
Dim retval As Long  ' return value

    retval = SetTimer(Form1.hWnd, 1, 500, AddressOf TimerProc)

End Sub

Private Sub Form1_Unload(Cancel As Integer)
    Dim retval As Long  ' return value

        retval = KillTimer(Form1.hWnd, 1)

    End Sub

See Also

SetTimer

Category

Timers

Back to the Reference section.

Back to the Function list.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/k/killtimer.html

LineTo Function

Declare Function LineTo Lib "gdi32.dll" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

LineTo draws a line from the current point to the point specified on a device. The line is drawn in the color specified by that object's .ForeColor property. After the line is drawn, the endpoint is the new current point. The algorithm Windows uses to draw a line does not actually color the last pixel of the line because it is not considered part of the line. The function returns 0 if an error occured, or 1 if successful.

hdc
The device context of the device to draw on.

x
The x coordinate of the endpoint to draw to.

y
The y coordinate of the endpoint to draw to.

Example:

' Draw a red line from (0,40) to (100,50) on the window Form1.
Dim pt As POINT_TYPE  ' needed for another API function
Dim retval As Long  ' return value

Form1.ForeColor = RGB(255, 0, 0)  ' set the foreground drawing color of Form1 to red
retval = MoveToEx(Form1.hdc, 0, 40, pt)  ' set the current point to (0,40)
retval = LineTo(Form1.hdc, 100, 50)  ' draw a line from current point to (100,50)

See Also: Polyline, PolylineTo, PolyPolyline

Category: Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lineto.html
LoadCursor Function

Declare Function LoadCursor Lib "user32.dll" Alias "LoadCursorA" (ByVal hInstance As Long, ByVal lpCursorName As Any) As Long

Platforms: Win 32s, Win 95/98, Win NT

LoadCursor loads a cursor from either a currently running program's cursor resources or Windows's cursor resources. The cursor can be referenced either by its resource name or by its numeric resource ID number. If successful, the function returns a handle to the loaded cursor. If unsuccessful, the function returns 0.

hInstance
To load one of a program's cursor resources, set this to the application's instance handle. To load one of Windows's cursor resources, set this to 0.

lpCursorName
Either a string containing the name of the cursor resource to load, or a numeric ID number identifying the resource. For Windows's cursors, exactly one of the following flags can be used to select the desired cursor resource:

IDC_APPSTARTING = 32650
The application starting cursor (arrow and hourglass).

IDC_ARROW = 32512
The regular arrow pointer cursor.

IDC_CROSS = 32515
The cross cursor.

IDC_IBEAM = 32513
The I-shaped beam cursor (text editing cursor).

IDC_ICON = 32641
Win NT only: An empty cursor.

IDC_NO = 32648
The "no" symbol cursor (circle with a slash).

IDC_SIZE = 32640
Win NT only: The four-pointed resize/move arrow.

IDC_SIZEALL = 32646
The four-pointed resize/move arrow.

IDC_SIZENESW = 32643
The double-pointed resize arrow pointing to the upper-right and lower-left.

IDC_SIZENS = 32645
The double-pointed resize arrow pointing up and down.

IDC_SIZENWSE = 32642
The double-pointed resize arrow pointing to the upper-left and lower-right.
The double-pointed resize arrow pointing left and right.
IDC_UPARROW = 32516
The up-arrow cursor.
IDC_WAIT = 32514
The wait cursor (hourglass).

Example:

' Display the application starting (arrow and hourglass) Windows
cursor for three seconds. The cursor resource is loaded from Windows. Then
' restore the old cursor (whatever it happens to be).
Dim hcursor As Long  ' receives handle to application starting cursor
Dim holdcursor As Long  ' receives handle to previously used cursor
Dim retval As Long  ' throw-away return value

cursor = LoadCursor(0, IDC_APPSTARTING)  ' load Windows's application starting
cursor
holdcursor = SetCursor(hcursor)  ' set it to the new cursor
Sleep 3000  ' wait for 3 seconds
retval = SetCursor(holdcursor)  ' set it to the previous cursor

See Also: LoadCursorFromFile
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
LoadCursorFromFile Function

Declare Function LoadCursorFromFile Lib "user32.dll" Alias "LoadCursorFromFileA"
(ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

LoadCursorFromFile loads a cursor from a cursor file. The cursor file can contain either a regular cursor (*.cur) or an animated cursor (*.ani). If successful, the function returns a cursor handle to the newly loaded cursor. If unsuccessful, the function returns 0.

lpFileName

The filename of the cursor file to load. This file can either be a *.cur or an *.ani cursor file.

Example:

' Load the cursor "C:\MyProg\custom.ani" and set it as
' the current cursor for three seconds. Then restore the original cursor as
' the current cursor.
Dim hcursor As Long  ' receives handle to the loaded cursor
Dim holdcursor As Long  ' receives handle to the previously in use cursor
Dim retval As Long  ' throw-away return value

hcursor = LoadCursorFromFile("C:\MyProg\custom.ani")  ' load the animated cursor from the file
If hcursor = 0 Then End  ' abort program if cursor couldn't be loaded
holdcursor = SetCursor(hcursor)  ' set the loaded cursor as the current cursor
Sleep 3000  ' wait for three seconds
retval = SetCursor(holdcursor)  ' restore the previous cursor

See Also: LoadCursor

Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
LocalFileTimeToFileTime Function

Declare Function LocalFileTimeToFileTime Lib "kernel32.dll" (lpLocalFileTime As FILETIME, lpFileTime As FILETIME) As Long

Platforms: Win 32s, Win 95/98, Win NT

LocalFileTimeToFileTime converts a time from local time (time according to the computer's current time zone) to UTC time (also known as Greenwich Mean Time). The source and target times are stored in FILETIME format. The function returns 1 if successful, or 0 if an error occured.

lpLocalFileTime
The time and date in local time to convert.

lpFileTime
Receives the time and date specified in lpLocalFileTime converted to UTC time.

Example:

' Convert the time and date May 1, 1999 6:10:00 PM local time
' to a FILETIME structure in UTC time.
Dim sourcetime As SYSTEMTIME  ' original time and date
Dim localtime As FILETIME  ' receives sourcetime's time
Dim utctime As FILETIME  ' receives the final result
Dim retval As Long  ' return value

' Set sourcetime to the desired date:
sourcetime.wMonth = 5: sourcetime.wDay = 1: sourcetime.wYear = 1999
sourcetime.wHour = 18: sourcetime.wMinute = 10: sourcetime.wSecond = 0

' Convert sourcetime into FILETIME format:
retval = SystemTimeToFileTime(sourcetime, localtime)
' Convert localtime into UTC time:
retval = LocalFileTimeToFileTime(localtime, utctime)
' utctime now has the converted time and date
See Also: FileTimeToLocalFileTime
Category: Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.
LockWorkStation Function

Declare Function LockWorkStation Lib "user32.dll" () As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Not Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Description & Usage

LockWorkStation locks the computer, preventing anybody from entering input until either the user who locked it or an administrator enters his or her password. This function does the same thing as though the user had pressed Ctrl+Alt+Del and selected "Lock Workstation."

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

None.

Example
Lock the workstation when the user clicks the command button named cmdLock. Obviously, to use this example, you must place a command button named cmdLock on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function LockWorkStation Lib "user32.dll" () As Long

' *** Place the following code inside the form window. ***

Private Sub cmdLock_Click()
    ' Lock the workstation when this button is clicked.
    Dim retval As Long ' return value

    retval = LockWorkStation()
    ' That's all there is to it!
End Sub

Category

Shutdown

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lockworkstation.html
Istrcmp Function

Declare Function lstrcmp Lib "kernel32.dll" Alias "lstrcmpA" (ByVal lpString1 As String, ByVal lpString2 As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

*Istrcmp* compares two strings using a case-sensitive comparison based on the current user locale. The function uses a word sort method, which places all symbols except hyphens and apostrophes before the letter "a" (hyphens and apostrophes are treated differently). The strings are compared by comparing the first character of each string, then the second character, etc., until unequal characters are encountered.

Return Value

If the function returns a negative value, the first string is less than the second string (i.e., the first string comes before the second string in alphabetical order). If the function returns 0, the two strings are equal. If the function returns a positive value, the first string is greater than the second string. In any case, the actual return value is the difference of the first unequal characters encountered.

Visual Basic-Specific Issues

None.

Parameters

*lpString1*
The first string to compare.

\textit{lpString2}

The second string to compare.

\section*{Example}

'This code is licensed according to the terms and conditions listed \url{here}.

'Use a case-sensitive comparison method to alphabetically sort
'nine words. The sorting method simply compares each possible pair
'of words; if a pair is out of alphabetical order, they are switched.

Dim words(1 To 9) As String ' the words to sort
Dim tempstr As String ' buffer used to swap strings
Dim oc As Integer, ic As Integer ' counter variables
Dim compval As Long ' result of comparison

' Load the nine strings into the array.
words(1) = "can't"
words(2) = "cant"
words(3) = "cannot"
words(4) = "pants"
words(5) = "co-op"
words(6) = "coop"
words(7) = "Denver"
words(8) = "denver"
words(9) = "denveR"

'Sort the strings, swapping any pairs which are out of order.
For oc = 1 To 8 ' first string of the pair
    For ic = oc + 1 To 9 ' second string of the pair
        ' Compare the two strings.
        compval = \texttt{lstrcmp(words(oc), words(ic))}
        ' If words(oc) is greater, swap them.
        If compval > 0 Then
            tempstr = words(oc)
            words(oc) = words(ic)
            words(ic) = tempstr
        End If
    Next ic
Next oc

'Display the list of sorted words.
For oc = 1 To 9
    Debug.Print words(oc)
Next oc
See Also

CompareString, lstrcmpi

Category

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.
Istrcmpi Function

Declare Function lstrcmpi Lib "kernel32.dll" Alias "lstrcmpiA" (ByVal lpString1 As String, ByVal lpString2 As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

Istrcmpi compares two strings using a case-insensitive comparison based on the current user locale. The function uses a word sort method, which places all symbols except hyphens and apostrophes before the letter "a" (hyphens and apostrophes are treated differently). The strings are compared by comparing the first character of each string, then the second character, etc., until unequal characters are encountered.

Return Value

If the function returns a negative value, the first string is less than the second string (i.e., the first string comes before the second string in alphabetical order). If the function returns 0, the two strings are equal. If the function returns a positive value, the first string is greater than the second string. In any case, the actual return value is the difference of the first unequal characters encountered.

Visual Basic-Specific Issues

None.

Parameters

lpString1
The first string to compare.

`lpString2`
The second string to compare.

## Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/l/lstrcmpi.html).

' Use a case-insensitive comparison method to alphabetically sort
' nine words. The sorting method simply compares each possible pair
' of words; if a pair is out of alphabetical order, they are switched.
' (Note how this sort will seemingly arrange the "Denver" trio in
' a random order, depending on how the search loops play out --
' the three strings are equal in the eyes of the function and therefore
' not sorted relative to each other.)
Dim words(1 To 9) As String  ' the words to sort
Dim tempstr As String  ' buffer used to swap strings
Dim oc As Integer, ic As Integer  ' counter variables
Dim compval As Long  ' result of comparison

' Load the nine strings into the array.
words(1) = "can't"
words(2) = "cant"
words(3) = "cannot"
words(4) = "pants"
words(5) = "co-op"
words(6) = "coop"
words(7) = "Denver"
words(8) = "denver"
words(9) = "denveR"

' Sort the strings, swapping any pairs which are out of order.
For oc = 1 To 8  ' first string of the pair
    For ic = oc + 1 To 9  ' second string of the pair
        ' Compare the two strings.
        compval = lstrcmpi(words(oc), words(ic))
        ' If words(oc) is greater, swap them.
        If compval > 0 Then
            tempstr = words(oc)
            words(oc) = words(ic)
            words(ic) = tempstr
        End If
    Next ic
Next oc
' Display the list of sorted words.
For oc = 1 To 9
  Debug.Print words(oc)
Next oc

See Also

CompareString, lstrcmp

Category

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: December 30, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lstrcmpi.html
Istrcpy Function

Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

Istrcpy copies the entire contents of one string into another string. Either string, instead of being a "real" string, can also be merely a pointer to a string instead. The target string must already have enough space to receive the source string's contents. The function also will copy a terminating null character into the target string.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

This function is very useful for "converting" a pointer to a string into an actual string.

Parameters

- lpString1
  String that receives the copied contents of lpString2. This could be either the string itself or a pointer to the string.
- lpString2
  Either an actual string to copy into lpString1 or a pointer to the string to copy into lpString1.
Example

' This code is licensed according to the terms and conditions listed [here.]

' Copy the source string to the target string
Dim source As String, target As String  ' the two strings
Dim retval As Long  ' return value

source = "Hello, world!"  ' the source string to copy
target = Space(Len(source))  ' make room in target to receive the copied string
retval = lstrcpy(target, source)  ' set target to equal source
Debug.Print "Source string: "; source
Debug.Print "Target string: "; target  ' they should be the same....

See Also

lstrcpyn

Category

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: December 22, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information] Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/l/lstrcpy.html]
Istrncpy Function

Declare Function lstrncpy Lib "kernel32.dll" Alias "lstrncpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any, ByVal iMaxLength As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

Istrncpy copies one or more characters from one string into another string, followed by a terminating null character. Either string, instead of being a "real" string, can also be merely a pointer to a string instead. The target string must already have enough space to receive the source string's contents along with the terminating null.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

This function is very useful for "converting" a pointer to a string into an actual string.

Parameters

- **lpString1**
  
  String that receives the copied contents of lpString2. This could be either the string itself or a pointer to the string.

- **lpString2**
  
  Either an actual string to copy into lpString1 or a pointer to the string to copy into lpString1.

- **iMaxLength**
  
  The maximum length of the target string.


The number of characters to copy from \textit{lpString2} to \textit{lpString1}, including the terminating null character added to the end. (For example, a value of 4 for this parameter would copy three characters from \textit{lpString2} and a null character into \textit{lpString1}.

\section*{Example}

\begin{verbatim}
' This code is licensed according to the terms and conditions listed here.

' Copy the first word source string to the target string
Dim source As String, target As String  ' the two strings
Dim retval As Long  ' return value

source = "Hello, world!"  ' the source string to copy
target = Space(6)  ' make room in target to receive the copied string
retval = lstrcpy(target, source, 6)  ' set target to equal source
target = Left(target, Len(target) - 1)  ' remove the terminating null character
Debug.Print "Source string: "; source
Debug.Print "Target string: "; target  ' this should be "Hello"
\end{verbatim}

\section*{See Also}

lstrcpy

\section*{Category}

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

\section*{Last Modified:} December 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lstrcpyn.html
Istrlen Function

Declare Function Istrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

Istrlen determines the length of a string. The length of a string is considered to be the number of characters it contains, not counting any possible terminating null character. This function can also determine the length of a string to which a pointer refers.

Return Value

The function returns the length of the string, measured in number of characters.

Visual Basic-Specific Issues

This function is very useful, in conjunction with Istrcpy, for "converting" a pointer to a string into an actual string.

Parameters

lpString

Either the string to determine the length of or a pointer to the string to determine the length of.
Example

' This code is licensed according to the terms and conditions listed here.
' Display the length of the string "Hello, world!"
Dim slength As Long ' receives the length of the string
slength = lstrlen("Hello, world!") ' find the length of the string
Debug.Print "The string 'Hello, world!' contains"; slength; "characters."

Category

Strings

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: December 21, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lstrlen.html
mciGetErrorString Function

Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

mciGetErrorString obtains a textual description of an error raised by another Media Control Interface (MCI) function. Typically these errors are not the fault of the program. Rather, they are caused by "problems" with the device (for example, the MIDI driver is currently being used by another program, so your program’s attempt to open it failed). The messages retrieved by this function are sufficient to tell the user what caused the error.

Return Value

If successful, the function returns a nonzero value. If an error occurred, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

- **fdwError**: A MCI error code returned by another MCI function. mciGetErrorString gets a description of this error.
- **lpszErrorText**: A string that receives a textual description of the error terminated by a null character. This string must already contain enough room to receive the text. This string should be at least 128 characters long.
- **cchErrorText**: The length of the string passed as lpszErrorText.

Example
To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
lpszCommand As String, ByVal lpszReturnString As String, ByVal cchReturnLength _
As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file. The file C:\Music\canyon.mid
' is opened when the form opens. The Play and Stop buttons behave as you'd expect.
The ' only potential surprise is that the current position is not reset when playback
' stops; it
' behaves just as pausing playback would. The file closes when the form unloads.

' If anything goes wrong in the example, display a message box with
' the MCI error message text.

Private Sub Form_Load()
    ' Open the file "C:\Music\canyon.mid" for later use in the example.
    ' Give it an alias of "canyon" so we don't need to refer to the filename
    ' again.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
    ' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
    ' Stop playback of the MIDI file when this button is pressed.
    ' The position within the file does not move back to the beginning.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads. This is important, because the ' MIDI driver can only work with one file at a time. There's no need to check
    ' for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error. There's
    ' no reason to use the MessageBox API function; VB's MsgBox function will suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long    ' return value
    ' Get a string explaining the MCI error.
    errstr = Space(128)
    retval = mciGetErrorString(errcode, errstr, Len(errstr))
    ' Remove the terminating null and empty space at the end.
    errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)
    ' Display a simple error message box.
    retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also
mciSendString

Category
Media Control Interface (MCI)

Back to the Function list.
Back to the Reference section.

---

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/mcigeterrorstring.html
mciSendString Function

Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal lpszCommand As String, ByVal lpszReturnString As String, ByVal cchReturnLength As Long, ByVal hwndCallback As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

mciSendString sends a command to a Multimedia Control Interface (MCI) device. The command strings used with this function can perform almost any task necessary for using a multimedia device installed on the computer. mciSendString provides a relatively easy way to perform multimedia output operations.

Look at the MCI command strings page for a list of some of the command strings used by the mciSendString function.

Return Value

If successful, the function returns 0. If an error occurred, the function returns a nonzero MCI error code. To get a textual description of the error, use the mciGetErrorString function.

Visual Basic-Specific Issues

None.

Parameters

_lpszCommand_

The command string to execute, including all of its necessary parameters and desired options.

_lpszReturnString_

For command strings that return information, this string receives the data output by the command. This string must initially be at least 128 characters long in order to receive the string. Any information placed into this string will be null-terminated. If the command string does not return any information, this parameter is ignored.

_cchReturn_

http://216.26.168.92/vbapi/ref/m/mcisendstring.html (1 of 4) [9/1/2002 5:35:30 PM]
The length of the string passed as `lpszReturnString`.

**hwndCallback**

If the "notify" flag of the command string is specified, this is a **handle** to the window to receive a **MM_MCINOTIFY** message when the command has completed, no matter whether it succeeded or failed.

**Example**

To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

```
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpszCommand As String, ByVal lpszReturnString As String, ByVal
    cchReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
    fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file.  The file C:\Music\canyon.mid
' is opened when the form opens.  The Play and Stop buttons behave as you'd expect.
' The ' only potential surprise is that the current position is not reset when playback
' stops; it ' behaves just as pausing playback would.  The file closes when the form unloads.
' If anything goes wrong in the example, display a message box with
' the MCI error message text.

Private Sub Form_Load()
    ' Open the file "C:\Music\canyon.mid" for later use in the example.
    ' Give it an alias of "canyon" so we don't need to refer to the filename again.
    Dim errcode As Long   ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
    ' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long   ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
```
' Stop playback of the MIDI file when this button is pressed.  
' The position within the file does not move back to the beginning.
Dim errcode As Long  ' MCI error code

errcode = mciSendString("stop canyon", ",", 0, 0)
If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
' Close the MIDI file when the form unloads.  This is important, because the 
' MIDI driver can only work with one file at a time.  There's no need to 
check
' for an error here, since we're just closing the file.
Dim errcode As Long  ' MCI error code

errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
' This subroutine displays a dialog box with the text of the MCI error. 
There's 
' no reason to use the MessageBox API function; VB's MsgBox function will 
suffice.
Dim errstr As String  ' MCI error message text
Dim retval As Long    ' return value

' Get a string explaining the MCI error.
erstr = Space(128)
retval = mciGetErrorString(errcode, errstr, Len(errstr))
' Remove the terminating null and empty space at the end.
erstr = Left(errstr, InStr(errstr, vbNullChar) - 1)

' Display a simple error message box.
retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also

mciGetErrorString

Category

Media Control Interface (MCI)

Back to the Function list.
Back to the Reference section.

Last Modified: July 4, 2000
MessageBeep Function

Declare Function MessageBeep Lib "user32.dll" (ByVal wType As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

MessageBeep plays one of the system's associated sounds, with one exception. These sounds are some of the ones that Windows associates with certain events. However, this function can also play a beep on the computer's internal speaker. The function returns 1 if successful, or 0 if an error occurred.

wType

If set to -1 (or &FFFFFFFF), plays a beep using the computer's internal speaker. Otherwise, this is exactly one of the following flags specifying which sound to play:

- MB_ICONASTERISK = &H40
  Play the SystemAsterisk sound.
- MB_ICONEXCLAMATION = &H30
  Play the SystemExclamation sound.
- MB_ICONHAND = &H10
  Play the SystemHand sound.
- MB_ICONQUESTION = &H20
  Play the SystemQuestion sound.
- MB_OK = &H0
  Play the SystemDefault sound.

Example:

' Play the SystemQuestion sound.
Dim retval As Long  ' return value
retval = MessageBeep(MB_ICONQUESTION)  ' play the SystemQuestion sound

See Also: Beep
Category: Errors
MessageBox Function

Declare Function MessageBox Lib "user32.dll" Alias "MessageBoxA" (ByVal hWnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal uType As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

*MessageBox* opens and operates a small message box on the screen. Message boxes are typically used either to communicate important information (such as an error message) or to prompt the user. The message box only closes when the user presses one of the buttons presented.

Return Value

If an error occurred, the function returns 0 (use *GetLastError* to get the error code). If successful, the function returns one of the following flags specifying the button the user clicked on:

IDABORT
   The Abort button was clicked.
IDCANCEL
   The Cancel button was clicked (or the user dismissed the message box using the Esc key).
IDCONTINUE
   **Windows 2000**: The Continue button was clicked.
IDIGNORE
   The Ignore button was clicked.
IDNO
   The No button was clicked.
IDOK
   The OK button was clicked.
IDRETRY
   The Retry button was clicked.
IDTRYAGAIN

**Windows 2000:** The Try Again button was clicked.

**IDYES**

The Yes button was clicked.

### Visual Basic-Specific Issues

None.

### Parameters

**hWnd**

A [handle](http://216.26.168.92/vbapi/ref/m/messagebox.html) to the window opening the message box.

**lpText**

The text to display inside the message box.

**lpCaption**

The text to display inside the caption area of the message box's title bar.

**uType**

A combination of various flags specifying the behavior and appearance of the message box. The available flags are grouped according to function. If no flags in a certain group are specified, the default is used.

Use one of the following flags to specify which buttons to display in the message box. (Note that MB_HELP can be combined with any of the other flags.)

- **MB_ABORTRETRYIGNORE**
  
  The message box contains the Abort, Retry, and Ignore buttons.

- **MBCancelarTRYCONTINUE**
  
  **Windows 2000:** The message box contains the Cancel, Try Again, and Continue buttons. This is meant to replace the MB_ABORTRETRYIGNORE flag.

- **MB_HELP**
  
  **Windows 95, 98, NT 4.0 or later, 2000:** Add the Help button to the message box. When the user clicks the Help button, the [WM_HELP](http://216.26.168.92/vbapi/ref/m/messagebox.html) message is sent to the owner of the message box (specified by the hWnd parameter). This flag can only be used by combining it with another button flag (i.e., the Help button cannot appear alone).

- **MB_OK**
  
  The message box contains the OK button. This is the default.

- **MB_OKCANCEL**
  
  The message box contains the OK and Cancel buttons.

- **MB_RETRYCANCEL**
  
  The message box contains the Retry and Cancel buttons.

- **MB_YESNO**
  
  The message box contains the Yes and No buttons.

- **MB_YENOCANCEL**
  
  The message box contains the Yes, No, and Cancel buttons.

Use one of the following flags to specify which icon to display in the message box:

- **MB_ICONASTERISK, MB_ICONINFORMATION**
  
  Display the information icon: a lowercase letter "i" inside a blue circle.

- **MB_ICONERROR, MB_ICONHAND, MB_ICONSTOP**
  
  Display the stop-sign icon in the message box.
MB_ICONEXCLAMATION, MB_ICONWARNING
    Display the exclamation-point icon in the message box.

MB_ICONQUESTION
    Display the question-mark icon in the message box.

Use one of the following flags to specify which button is selected by default:
MB_DEFBUTTON1
    The first button is the default. This is the default.

MB_DEFBUTTON2
    The second button is the default.

MB_DEFBUTTON3
    The third button is the default.

MB_DEFBUTTON4
    The fourth button is the default.

Use one of the following flags to specify the modality of the message box:
MB_APPLMODAL
    The message box is application-modal. The user cannot switch to any other windows owned by
    the application until he or she first closes the message box. This is the default.

MB_SYSTEMMODAL
    The message box is system-modal. The user cannot switch to any other windows until he or she first
    closes the message box.

MB_TASKMODAL
    The message box is thread-modal. The user cannot switch to any other windows owned by the calling thread
    until he or she first closes the message box.

Use zero or more of the following flags to specify other options for the message box:
MB_DEFAULT_DESKTOP_ONLY
    Windows NT, 2000: Same as MB_SERVICE_NOTIFICATION, except that the system will display the
    message box only on the interactive window station's default desktop.

MB_RIGHT
    The text in the message box is right-justified.

MB_RTLREADING
    Display the message text and caption using right-to-left reading order if desired by the system language.

MB_SETFOREGROUND
    Make the message box the foreground window.

MB_TOPMOST
    Make the message box a topmost window.

MB_SERVICE_NOTIFICATION
    Windows NT 4.0 or later, 2000: The calling thread is a service notifying the user of an event. The hWnd
    parameter must be 0.

MB_SERVICE_NOTIFICATION_NT3X
    Windows NT 3.1 through 3.51: Same as MB_SERVICE_NOTIFICATION. The value of this flag changed
    with the release of NT 4.0.

## Constant Definitions

Const IDABORT = 3
Const IDCANCEL = 2
Const IDCONTINUE = 5
Const IDIGNORE = 5
Const IDNO = 7
Const IDOK = 1
Const IDRETRY = 4
Const IDTRYAGAIN = 4
Const IDYES = 6
Const MB_ABORTRETRYIGNORE = &H2
Const MB_CANCELTRYCONTINUE = &H2
Const MB_HELP = &H4000
Const MB_OK = &H0
Const MB_OKCANCEL = &H1
Const MB_RETRYCANCEL = &H5
Const MB_YESNO = &H4
Const MB_YESNOCANCEL = &H3
Const MB_ICONASTERISK = &H40
Const MB_ICONERROR = &H10
Const MB_ICONEXCLAMATION = &H30
Const MB_ICONHAND = &H10
Const MB_ICONINFORMATION = &H40
Const MB_ICONQUESTION = &H20
Const MB_ICONSTOP = &H10
Const MB_ICONWARNING = &H30
Const MB_DEFBUTTON1 = &H0
Const MB_DEFBUTTON2 = &H100
Const MB_DEFBUTTON3 = &H200
Const MB_DEFBUTTON4 = &H300
Const MB_APPLMODAL = &H0
Const MB_SYSTEMMODAL = &H1000
Const MB_TASKMODAL = &H2000
Const MB_DEFAULT_DESKTOP_ONLY = &H20000
Const MB_RIGHT = &H80000
Const MB_RTLREADING = &H100000
Const MB_SETFOREGROUND = &H10000
' Const MB_TOPMOST = ???
' Const MB_SERVICE_NOTIFICATION = ???
' Const MB_SERVICE_NOTIFICATION_NT3X = ???

Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/m/messagebox.html).

' Display a warning message box with the Yes and No option
' owned by window Form1. Make the message box system-modal to force the
' user to reply immediately. Have "No" selected by default.
Dim mbresult As Long ' result of message box
Dim flags As Long ' message box flags

' For convenience, use a variable to represent the flag settings.
flags = MB_YESNO Or MB_ICONWARNING Or MB_DEFBUTTON2 Or MB_SYSTEMMODAL
' Display the message box.
mbresult = MessageBox(Form1.hWnd, "Are you sure you want to do that?", "Warning!", flags)
' Determine what the user pressed.
If mbresult = IDYES Then
    Debug.Print "OK, go ahead and do that."
Elseif mbresult = IDNO Then
    Debug.Print "Operation aborted."
End If

See Also

MessageBoxEx, MessageBoxIndirect

Category

Dialog Boxes

Back to the Function list.
Back to the Reference section.

Last Modified: January 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/messagebox.html
MessageBoxEx Function

Declare Function MessageBoxEx Lib "user32.dll" Alias "MessageBoxExA" (ByVal hWnd As Long, ByVal lpText As String, ByVal lpCaption As String, ByVal uType As Long, ByVal wLanguageId As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

MessageBoxEx opens and operates a small message box on the screen. Message boxes are typically used either to communicate important information (such as an error message) or to prompt the user. The message box only closes when the user presses one of the buttons presented.

Return Value

If an error occured, the function returns 0 (use GetLastError to get the error code). If successful, the function returns one of the following flags specifying the button the user clicked on:

- IDABORT: The Abort button was clicked.
- IDCANCEL: The Cancel button was clicked (or the user dismissed the message box using the Esc key).
- IDCONTINUE: Windows 2000: The Continue button was clicked.
- IDIGNORE: The Ignore button was clicked.
- IDNO: The No button was clicked.
- IDOK: The OK button was clicked.
- IDRETRY: The Retry button was clicked.
- IDTRYAGAIN:
**Windows 2000:** The Try Again button was clicked.

**IDYES**

The Yes button was clicked.

## Visual Basic-Specific Issues

None.

## Parameters

**hWnd**

A *handle* to the window opening the message box.

**lpText**

The text to display inside the message box.

**lpCaption**

The text to display inside the caption area of the message box's title bar.

**uType**

A combination of various flags specifying the behavior and appearance of the message box. The available flags are grouped according to function. If no flags in a certain group are specified, the default is used.

Use one of the following flags to specify which buttons to display in the message box. (Note that `MB_HELP` can be combined with any of the other flags.)

- **MB_ABORTRETRYIGNORE**: The message box contains the Abort, Retry, and Ignore buttons.
- **MB_CANCELTRYCONTINUE**: *Windows 2000:* The message box contains the Cancel, Try Again, and Continue buttons. This is meant to replace the `MB_ABORTRETRYIGNORE` flag.
- **MB_HELP**: *Windows 95, 98, NT 4.0 or later, 2000:* Add the Help button to the message box. When the user clicks the Help button, the `WM_HELP` message is sent to the owner of the message box (specified by the `hWnd` parameter). This flag can only be used by combining it with another button flag (i.e., the Help button cannot appear alone).
- **MB_OK**: The message box contains the OK button. This is the default.
- **MB_OKCANCEL**: The message box contains the OK and Cancel buttons.
- **MB_RETRYCANCEL**: The message box contains the Retry and Cancel buttons.
- **MB_YESNO**: The message box contains the Yes and No buttons.
- **MB_YENOCANCEL**: The message box contains the Yes, No, and Cancel buttons.

Use one of the following flags to specify which icon to display in the message box:

- **MB_ICONASTERISK, MB_ICONINFORMATION**: Display the information icon: a lowercase letter "i" inside a blue circle.
- **MB_ICONERROR, MB_ICONHAND, MB_ICONSTOP**: Display the stop-sign icon in the message box.
Windows API Guide: MessageBoxEx Function

MB_ICONEXCLAMATION, MB_ICONWARNING
Display the exclamation-point icon in the message box.

MB_ICONQUESTION
Display the question-mark icon in the message box.

Use one of the following flags to specify which button is selected by default:
MB_DEFBUTTON1
The first button is the default. This is the default.

MB_DEFBUTTON2
The second button is the default.

MB_DEFBUTTON3
The third button is the default.

MB_DEFBUTTON4
The fourth button is the default.

Use one of the following flags to specify the modality of the message box:
MB_APPLMODAL
The message box is application-modal. The user cannot switch to any other windows owned by the application until he or she first closes the message box. This is the default.

MB_SYSTEMMODAL
The message box is system-modal. The user cannot switch to any other windows until he or she first closes the message box.

MB_TASKMODAL
The message box is thread-modal. The user cannot switch to any other windows owned by the calling thread until he or she first closes the message box.

Use zero or more of the following flags to specify other options for the message box:
MB_DEFAULT_DESKTOP_ONLY
Windows NT, 2000: Same as MB_SERVICE_NOTIFICATION, except that the system will display the message box only on the interactive window station's default desktop.

MB_RIGHT
The text in the message box is right-justified.

MB_RTLREADING
Display the message text and caption using right-to-left reading order if desired by the system language.

MB_SETFOREGROUND
Make the message box the foreground window.

MB_TOPMOST
Make the message box a topmost window.

MB_SERVICE_NOTIFICATION
Windows NT 4.0 or later, 2000: The calling thread is a service notifying the user of an event. The hWnd parameter must be 0.

MB_SERVICE_NOTIFICATION_NT3X
Windows NT 3.1 through 3.51: Same as MB_SERVICE_NOTIFICATION. The value of this flag changed with the release of NT 4.0.

wLanguageId
A language identifier, created by MAKELANGID, which identifies the language used to display the message box's buttons. The specified language must be installed on the system.

Constant Definitions
Windows API Guide: MessageBoxEx Function

Const IDABORT = 3
Const IDCANCEL = 2
Const IDCONTINUE = 5
Const IDIGNORE = 5
Const IDNO = 7
Const IDOK = 1
Const IDRETRY = 4
Const IDTRYAGAIN = 4
Const IDYES = 6
Const MB_ABORTRETRYIGNORE = &H2
Const MB_CANCELTRYCONTINUE = &H2
Const MB_HELP = &H4000
Const MB_OK = &H0
Const MB_OKCANCEL = &H1
Const MB_RETRYCANCEL = &H5
Const MB_YESNO = &H4
Const MB_ICONASTERISK = &H40
Const MB_ICONERROR = &H10
Const MB_ICONEXCLAMATION = &H30
Const MB_ICONHAND = &H10
Const MB_ICONINFORMATION = &H40
Const MB_ICONQUESTION = &H20
Const MB_ICONSTOP = &H10
Const MB_ICONWARNING = &H30
Const MB_DEFBUTTON1 = &H0
Const MB_DEFBUTTON2 = &H100
Const MB_DEFBUTTON3 = &H200
Const MB_DEFBUTTON4 = &H300
Const MB_APPLMODAL = &H0
Const MB_SYSTEMMODAL = &H1000
Const MB_TASKMODAL = &H2000
Const MB_DEFAULT_DESKTOP_ONLY = &H20000
Const MB_RIGHT = &H80000
Const MB_RTLREADING = &H100000
Const MB_SETFOREGROUND = &H10000
'Const MB_TOPMOST = ???
'Const MB_SERVICE_NOTIFICATION = ???
'Const MB_SERVICE_NOTIFICATION_NT3X = ???

Example

' This code is licensed according to the terms and conditions listed here.

' Display a warning message box with the Yes and No option
' owned by window Form1. Make the message box system-modal to force the
' user to reply immediately. Have "No" selected by default, and display
' the buttons in American English.
Dim mbresult As Long ' result of message box
Dim flags As Long  ' message box flags
Dim lang As Long  ' language ID

' For convenience, use a variable to represent the flag settings.
flags = MB_YESNO Or MB_ICONWARNING Or MB_DEFBUTTON2 Or MB_SYSTEMMODAL
' Generate the proper language ID.
lang = MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US)
' Display the message box.
mbresult = MessageBox(Form1.hWnd, "Are you sure you want to do that?", "Warning!",
flags, lang)
' Determine what the user pressed.
If mbresult = IDYES Then
    Debug.Print "OK, go ahead and do that."
ElseIf mbresult = IDNO Then
    Debug.Print "Operation aborted."
End If

See Also

MessageBox, MessageBoxIndirect

Category

Dialog Boxes

Back to the Function list.
Back to the Reference section.

Last Modified: February 10, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/messageboxex.html
MessageBoxIndirect Function

Declare Function MessageBoxIndirect Lib "user32.dll" Alias "MessageBoxIndirectA" (lpMsgBoxParams As MSGBOXPARAMS) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

MessageBoxIndirect opens and operates a small message box on the screen. Message boxes are typically used either to communicate important information (such as an error message) or to prompt the user. This function offers more options for the creation of the message box than MessageBox and MessageBoxIconEx do because this function stores those options in a structure passed to the function. The message box only closes when the user presses one of the buttons presented.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns one of the following flags specifying the button the user clicked on:

IDABORT
   The Abort button was clicked.
IDCANCEL
   The Cancel button was clicked (or the user dismissed the message box using the Esc key).
IDCONTINUE
   Windows 2000: The Continue button was clicked.
IDIGNORE
   The Ignore button was clicked.
IDNO
   The No button was clicked.
IDOK
   The OK button was clicked.
IDRETRY
   The Retry button was clicked.
IDTRYAGAIN

Windows 2000: The Try Again button was clicked.

IDYES

The Yes button was clicked.

Visual Basic-Specific Issues

None.

Parameters

lpMsgBoxParams

The settings used to create and operate the message box.

Constant Definitions

Const IDABORT = 3
Const IDCANCEL = 2
Const IDCONTINUE = 5
Const IDIGNORE = 5
Const IDNO = 7
Const IDOK = 1
Const IDRETRY = 4
Const IDTRYAGAIN = 4
Const IDYES = 6

Example

' This code is licensed according to the terms and conditions listed here.

' Create a Yes/No dialog for the user. Also offer a Help button which opens the corresponding topic in a WinHelp file.
' The message box is owned by window Form1. Pay attention to where various parts of the example code must be placed.

' *** Place the following code in a module. ***
' This callback function handles the message box's Help button.
Public Sub MsgBoxCallback (lpHelpInfo As HELPINFO)
    Dim retval As Long ' return value

    ' Open the proper topic in the help file "C:\MyProg\proghelp.hlp".
    ' The desired Context ID is found inside the structure.
    retval = WinHelp(Form1.hWnd, "C:\MyProg\proghelp.hlp", HELP_CONTEXT, lpHelpInfo.dwContextId)
End Sub
This is a dummy function, merely returning what is passed to it. This is needed because the AddressOf operator is only valid inside a function call, whereas we need to use it to set a data member.

Public Function DummyFunc (ByVal param As Long) As Long
    DummyFunc = param
End Function

*** Place the following code where you want to create the message box. ***
Dim mbp As MSGBOXPARAMS ' message box's parameters
Dim mbresult As Long ' result of message box

' Fill in the parameter structure for the message box.
With mbp
    .cbSize = Len(mbp) ' size of structure
    .hwndOwner = Form1.hWnd ' window which owns the message box
    .hInstance = App.hInstance ' handle to instance of this program
    .lpszText = "Are you sure you want to do that?" ' message box text
    .lpszCaption = "User Prompt" ' message box's title bar text
    .dwStyle = MB_YESNO Or MB_HELP Or MB_ICONQUESTION ' flags for display
    .lpszIcon = 0 ' not needed
    .dwContextHelpId = 2300 ' Context ID for the proper topic in the Help file
    .lpfnMsgBoxCallback = DummyFunc(AddressOf MsgBoxCallback) ' pointer to callback function
    .dwLanguageId = MAKELANGID(LANG_ENGLISH, SUBLANG_ENGLISH_US) ' use American English
End With

' Display the message box.
mbresult = MessageBoxIndirect (mbp)
' Determine what the user pressed.
If mbresult = IDYES Then
    Debug.Print "OK, go ahead and do that."
ElseIf mbresult = IDNO Then
    Debug.Print "Operation aborted."
End If

See Also

MessageBox, MessageBoxEx

Category

Dialog Boxes

Back to the Function list.
Back to the Reference section.
Last Modified: February 11, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/messageboxindirect.html
mouse_event Function

Declare Sub mouse_event Lib "user32.dll" (ByVal dwFlags As Long, ByVal dx As Long, ByVal dy As Long, ByVal cButtons As Long, ByVal dwExtraInfo As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

mouse_event synthesizes mouse input by placing mouse input information into the input stream. A single mouse input event consists of either a move of the mouse or the change of the button state. For mouse movement, the coordinates can be given in either absolute or relative form. Only changes in mouse position or button state should be send via this function. For example, if the left mouse button is already down, the program should not send another left-button-down input.

Return Value

mouse_event does not return a value.

Visual Basic-Specific Issues

None.

Parameters

- dwFlags: A combination of the following flags specifying which mouse input information to place into the input stream. Remember to only specify button status information for those which have changed. Note that scroll wheel movement and X button status cannot be simultaneously specified because they both use the dwData parameter.
  - MOUSEEVENTF_ABSOLUTE: The dx and dy parameters contain absolute mouse coordinates. In the coordinate system used by the function, the screen's upper-left corner has coordinates (0,0) and the lower-right corner has coordinates (65535,65535), regardless of the actual screen size. If this flag is not set, dx and dy contain relative coordinates, whose actual amount of movement depends on the current mouse speed and acceleration settings.
MOUSEEVENTF_LEFTDOWN
   The left button was pressed.
MOUSEEVENTF_LEFTUP
   The left button was released.
MOUSEEVENTF_MIDDLEDOWN
   The middle button was pressed.
MOUSEEVENTF_MIDDLEUP
   The middle button was released.
MOUSEEVENTF_MOVE
   The mouse moved. The dx and dy parameters specify the amount or location of the movement.
MOUSEEVENTF_RIGHTDOWN
   The right button was pressed.
MOUSEEVENTF_RIGHTUP
   The right button was released.
MOUSEEVENTF_WHEEL
   Windows NT, 2000: The scroll wheel has moved. The dwData parameter specifies the amount of movement.
   Windows 2000: An X button was pressed. The dwData parameter identifies which X buttons.
MOUSEEVENTF_XDOWN
   Windows 2000: An X button was released. The dwData parameter identifies which X buttons.

dx
   Specifies either the x-coordinate of absolute mouse movement or the amount of relative movement along the x-axis. For relative motion, positive values move right and negative values move left.

dy
   Specifies either the y-coordinate of absolute mouse movement or the amount of relative movement along the y-axis. For relative motion, positive values move down and negative values move up.

dwData
   Windows NT, 2000: If dwFlags contains MOUSEEVENTF_WHEEL, this specifies the amount of wheel movement, in integer multiples of WHEEL_DELTA. Positive values mean forward (away) rotation, and negative values mean backwards (toward) rotation. Windows 2000: If dwFlags contains either MOUSEEVENTF_XDOWN or MOUSEEVENTF_XUP, this is a combination of the following flags specifying which X buttons have been pressed or released:
   XBUTTON1
   The first X button was pressed or released.
   XBUTTON2
   The second X button was pressed or released.

dwExtraInfo
   An additional 32-bit value associated with the mouse event.

Constant Definitions

Const MOUSEEVENTF_ABSOLUTE = &H8000
Const MOUSEEVENTF_LEFTDOWN = &H2
Const MOUSEEVENTF_LEFTUP = &H4
Const MOUSEEVENTF_MIDDLEDOWN = &H20
Const MOUSEEVENTF_MIDDLEUP = &H40
Const MOUSEEVENTF_MOVE = &H1
Const MOUSEEVENTF_RIGHTDOWN = &H8
Const MOUSEEVENTF_RIGHTUP = &H10
Const MOUSEEVENTF_WHEEL = &H80
Const MOUSEEVENTF_XDOWN = &H100
Const MOUSEEVENTF_XUP = &H200
Const WHEEL_DELTA = 120
Const XBUTTON1 = &H1
Const XBUTTON2 = &H2

Example

Simulate clicking the left mouse button at the upper-left corner of the screen. Although we could use the `mouse_event` function to move the cursor, it is much easier to use `SetCursorPos` instead. The example runs when the user clicks button Command1. Obviously, to use this example, you must place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Sub mouse_event Lib "user32.dll" (ByVal dwFlags As Long, ByVal dx As Long, ByVal dy As Long, ByVal cButtons As Long, ByVal dwExtraInfo As Long)
Public Declare Function SetCursorPos Lib "user32.dll" (ByVal x As Long, ByVal y As Long) As Long

Private Sub Command1_Click()
    Dim retval As Long ' return value
    
    ' Move the mouse cursor to the upper-left corner of the screen.
    retval = SetCursorPos(0, 0)
    ' Click the left mouse button once.
    mouse_event MOUSEEVENTF_LEFTDOWN, 0, 0, 0, 0
    mouse_event MOUSEEVENTF_LEFTUP, 0, 0, 0, 0

End Sub

See Also

`keybd_event`, `SendInput`

Category

Mouse

Go back to the Function listing.
Go back to the Reference section index.

---

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000

MoveFile Function

Declare Function MoveFile Lib "kernel32.dll" Alias "MoveFileA" (ByVal lpExistingFileName As String, ByVal lpNewFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

MoveFile moves or renames a file or directory -- it's really the same operation either way. If a directory is moved/renamed, all of the subdirectories and files contained in it will similarly be moved/renamed to reflect the path change. The function returns 1 if successful, or 0 if an error occured.

lpExistingFileName
The source file or directory; i.e., the file or directory to rename (move).

lpNewFileName
The target file or directory; i.e., the new file or directory name to give the source file (where to move the file or directory).

Example:

' Move the file to C:\MyFiles\temp.txt to C:\Dummy\buffer.txt.
' The original file will no longer exist. Note how this example both changes
' the filename and moves the file into a different directory simultaneously.
Dim retval As Long  ' return value

retval = MoveFile("C:\MyFiles\temp.txt", "C:\Dummy\buffer.txt")

See Also: CopyFile
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
MoveMemory Function

Declare Sub MoveMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

MoveMemory moves the contents of a portion of memory from one location to another. The two locations are identified by pointers to the memory addresses. After the copy, the original contents in the source are set to zeros.

Return Value

MoveMemory does not return a value.

Visual Basic-Specific Issues

A pointer to any variable can be automatically generated merely be passing that variable as either Destination or Source. However, if either a String or a Long holding the desired memory address is passed, the ByVal keyword must preceed it.

Parameters

\( \text{Destination} \)

A pointer to the memory address to use as the target, which receives the transfered data.

\( \text{Source} \)

A pointer to the memory address to use as the source, which initially holds the data to be transfered.

\( \text{Length} \)

The number of bytes of data to copy from the source memory location to the target memory location.

Example
This code is licensed according to the terms and conditions listed [here](http://www.vbapi.com/ref/m/movememory.html).

Transfer the contents of one byte array to another. After the transfer, the contents of the source array are set to 0.

```vba
Dim source(0 To 9) As Byte   ' source array of 10 bytes
Dim target(0 To 9) As Byte   ' similarly sized target array
Dim c As Integer            ' counter variable

' Fill the source array with some information.
For c = 0 To 9               ' loop through each element
    source(c) = c          ' set each element's value to its index
Next c

' Transfer the data from the target array to the source array. Note how pointers are implied merely by passing the arrays as usual.
MoveMemory target(0), source(0), 10   ' copy all 10 bytes

' Verify that the contents were transferred.
For c = 0 To 9
    Debug.Print target(c);  ' this will now contain the information
Next c
```

See Also

- CopyMemory

Category

- Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

Last Modified: July 28, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/m/movememory.html)
Go back to the Windows API Guide home page.
E-mail: [vbapi@vbapi.com](mailto:vbapi@vbapi.com) [Send Encrypted E-Mail](http://216.26.168.92/vbapi/ref/m/movememory.html) (2 of 2) [9/1/2002 5:36:21 PM]
MoveToEx Function

Declare Function MoveToEx Lib "gdi32.dll" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, lpPoint As POINT_TYPE) As Long

Platforms: Win 32s, Win 95/98, Win NT

MoveToEx sets the current point of a device. The current point is the starting point from which all graphics APIs ending with "To" (such as LineTo) begin drawing from. Some programming languages call this point the last point referenced. This function also puts the former current point into the variable passed as lpPoint. The function returns 0 if an error occurred, or 1 if successful.

hdc
   The device context of the device to set the current point of.

x
   The x coordinate of the point to set as the current point.

y
   The y coordinate of the point to set as the current point.

lpPoint
   Variable that receives the coordinate of the former current point.

Example:

' Draw a red line from (0,40) to (100,50) on the window Form1.
Dim pt As POINT_TYPE  ' receives the former current point
Dim retval As Long    ' return value
Form1.ForeColor = RGB(255, 0, 0)  ' set the foreground drawing color of Form1 to red
retval = MoveToEx(Form1.hdc, 0, 40, pt)  ' set the current point to (0,40)
' Note that pt now contains whatever the old current point was, but it doesn't matter here.
retval = LineTo(Form1.hdc, 100, 50)  ' draw a line from current point to (100,50)

Category: Lines & Curves
MoveWindow Function

Declare Function MoveWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal bRepaint As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

MoveWindow moves a window to a new location. In addition to moving it, this function also changes the window's size to a new width and height. The function returns 1 if successful, or 0 if an error occurred.

hwnd
The handle of the window to move and resize.

x
The x-coordinate to position the upper-left corner of the window at.

y
The y-coordinate to position the upper-left corner of the window at.

nWidth
The width in pixels to resize the window to.

nHeight
The height in pixels to resize the window to.

bRepaint
If 1, updates the screen to display the window at its new position. If 0, does not update the screen to reflect the move (the window will appear to be unmoved but will actually be at its new location!).

Example:

' Move window Form1. Set its upper-left corner to the point (200, 150).
' Change its size to a width of 175 and a height of 300.
Dim retval As Long  ' return value

' Move the window and make sure it's redrawn at its new position.
retval = MoveWindow(Form1.hWnd, 200, 150, 175, 300, 1)
' (If the last value had been 0, the window would have appeared to be unmoved!)

See Also: GetWindowRect, SetWindowPos

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
MulDiv Function

Declare Function MulDiv Lib "kernel32.dll" (ByVal nNumber As Long, ByVal nNumerator As Long, ByVal nDenominator As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

MulDiv multiplies two 32-bit integers together and divides the resulting 64-bit integer by a 32-bit integer. The final result is (usually) a 32-bit integer. This function is useful because it can calculate expressions which, although resulting in a perfectly acceptable value, would otherwise cause an overflow error if the programming language’s arithmetic operators were used. If the end result is not an integer, it is rounded to the nearest integer.

Return Value

If an error occurred (such as if the result is greater than a 32-bit value or division by zero was attempted), the function returns -1. If successful, the function returns the result of the expression \((nNumber \times nNumerator) / nDenominator\) rounded to the nearest integer.

Visual Basic-Specific Issues

None.

Parameters

nNumber
The first number to multiply.
nNumerator
The second number to multiply.
nDenominator
The number to divide by.

Example

' This code is licensed according to the terms and conditions listed here.
' Demonstrate how MulDiv can help avoid producing overflow errors.
Dim result As Long

' Without MulDiv: If you uncomment the line below, an overflow error ' will occur because the result of the multiplication is greater ' than a 32-bit value.
'result = -134217728 * 243 / 110592

' With MulDiv: There is no problem calculating the result.
result = MulDiv(-134217728, 243, 110592)
' (The result is exactly -294912.)

Category

Math

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 9, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/muldiv.html
OffsetRect Function

Declare Function OffsetRect Lib "user32.dll" (lpRect As RECT, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

OffsetRect changes the position of a rectangle without changing its size. If the value to move by is negative, the rectangle is moved left or up (depending on the direction); positive values move it right or down. The function returns 0 if an error occurred, or 1 if successful.

lpRect
The rectangle to move.

x
The value to move the rectangle horizontally by. Negative values move to the left, positive to the right.

y
The value to move the rectangle vertically by. Negative values move up, positive down.

Example:

' Shift window Form1 50 pixels right and 20 pixels up using its rectangle.
Dim winrect As RECT ' receives the rectangle of the window
Dim retval As Long ' return value

retval = GetWindowRect(Form1.hWnd, winrect) ' get Form1's rectangle
retval = OffsetRect(winrect, 50, -20) ' shift the rectangle 50 to the right and 20 upwards
' Now change the window on screen to match its new rectangle
retval = SetWindowPos(Form1.hWnd, 0, winrect.Left, winrect.Top, winrect.Right, winrect.Bottom, 0)

See Also: InflateRect
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
OffsetRgn Function

Declare Function OffsetRgn Lib "gdi32.dll" (ByVal hRgn As Long, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

OffsetRgn translates (slides) a region by a specified amount horizontally and vertically. The region can be moved in any direction left, right, up, or down. The function returns 0 if an error occurred, or exactly one of the following flags identifying the shape of the region which was moved:

COMPLEXREGION = 3
    The region is not empty but is not a rectangle.
NULLREGION = 1
    The region is empty, i.e., null.
SIMPLEREGION = 2
    The region is rectangular in shape.

hRgn
    A handle to the region to move.
x
    The number of pixels to move the region horizontally. Positive values move to the right; negative ones move to the left.
y
    The number of pixels to move the region vertically. Positive values move down; negative ones move up.

Example:

' On window Form1, fill an elliptical region in light gray. Then translate the region 50 pixels right and 20 pixels up and fill it with dark gray.
Dim hRgn As Long  ' handle to the region
Dim hLightBrush As Long, hDarkBrush As Long  ' handles to the two brushes to be used
Dim retval As Long  ' generic return value

' Create the elliptical region.
hRgn = CreateEllipticRgn(20, 100, 220, 200)  ' bounding rectangle (20,100)-(220,200)
' Get handles to the light and dark gray solid stock brushes.
hLightBrush = GetStockObject(LTGRAY_BRUSH)
hDarkBrush = GetStockObject(DKGRAY_BRUSH)

' Fill in the region in its current location on Form1 in light gray.
retval = FillRgn(Form1.hDC, hRgn, hLightBrush)
' Slide the region 50 pixels right and 20 pixels up.
retval = OffsetRgn(hRgn, 50, -20)  ' -20 means 20 up, not down
' Fill in the region in its new location in dark gray.
retval = FillRgn(Form1.hDC, hRgn, hDarkBrush)

' Delete the region to free up resources.
retval = DeleteObject(hRgn)

Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
OpenPrinter Function

Declare Function OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, pDefault As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

OpenPrinter opens a printer and obtains a handle to it. This handle can then be used by a number of printer API functions to reference the printer. After your program is finished using this handle, it should close the printer via ClosePrinter.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

When passing zero for pDefault, the expression ByVal CLng(0) must be used.

Parameters

pPrinterName
  The name of the printer or print server to obtain a handle to. If this is an empty string, the function obtains a handle to the default print server.

phPrinter
  Receives a handle to the newly opened printer.

pDefault
  A PRINTER_DEFAULTS structure specifying some options for opening the printer. To use the default settings, pass zero for this parameter.

Example
Display the Properties dialog box for the system's default printer. The dialog box is opened when the user clicks the button cmdProperties. To use the example, first place a command button named cmdProperties on a form window.

' This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type PRINTER_INFO_1
 dimensions: 612.0x792.0
[18x780]Public Type PRINTER_INFO_1
 flags As Long
 pDescription As String
 pName As String
 pComment As String
End Type

Public Declare Function EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal flags As Long, _
 ByVal name As String, ByVal Level As Long, pPrinterEnum As Long, ByVal cdBuf As Long, _
 pcbNeeded As Long, pcReturned As Long) As Long
Public Const PRINTER_ENUM_DEFAULT = &H1
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, _
 ByVal lpString2 As Any) As Long
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, pDefault As Any) As Long
Public Declare Function PrinterProperties Lib "winspool.drv" (ByVal hWnd As Long, ByVal hPrinter As Long) As Long
Public Declare Function ClosePrinter Lib "winspool.drv" (ByVal hPrinter As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub cmdProperties_Click()
    Dim pi1 As PRINTER_INFO_1 ' a little info about the printer
    Dim bytesNeeded As Long ' size needed for buffer
    Dim numPrinters As Long ' number of printers enumerated (should be 1)
    Dim buffer() As Long ' buffer for printer information
    Dim slength As Long ' length of string to copy
    Dim hPrinter As Long ' handle to the printer
    Dim retval As Long ' generic return value

    ' Figure out how much space is needed to store the printer information.
    retval = EnumPrinters(PRINTER_ENUM_DEFAULT, vbNullString, 1, ByVal 0, 0, bytesNeeded, numPrinters)
    ReDim buffer(0 To bytesNeeded / 4 - 1) As Long
    ' Get information about the default printer.
    retval = EnumPrinters(PRINTER_ENUM_DEFAULT, vbNullString, 1, buffer(0),
bytesNeeded, _
    bytesNeeded, numPrinters)
    ' Make sure we were successful.
    If retval = 0 Or numPrinters <> 1 Then
        Debug.Print "No default printer or some other error."
        Exit Sub
    End If

    ' Copy the data into the structure.
    With pi1
        ' Copy numerical data directly.
        .flags = buffer(0)
        ' Strings require more work, since the buffer holds pointers to them.
        .pDescription = Space(lstrlen(buffer(1)))
        retval = lstrcpy(.pDescription, buffer(1))
        .pName = Space(lstrlen(buffer(2)))
        retval = lstrcpy(.pName, buffer(2))
        .pComment = Space(lstrlen(buffer(3)))
        retval = lstrcpy(.pComment, buffer(3))
    End With

    ' Open the printer.
    retval = OpenPrinter(pi1.pName, hPrinter, ByVal CLng(0))
    If retval <> 0 Then
        ' Display the properties dialog.
        retval = PrinterProperties(Me.hWnd, hPrinter)
        ' Close the printer.
        retval = ClosePrinter(hPrinter)
    Else
        Debug.Print "Unable to open printer!"
    End If
End Sub

See Also

ClosePrinter

Category

Printers

Go back to the Function listing.
Go back to the Reference section index.
PickIconDlg Function

Declare Function PickIconDlg Lib "shell32.dll" Alias "#62" (ByVal hwndOwner As Long, ByVal lpstrFile As String, ByVal nMaxFile As Long, lpdwIconIndex As Long) As Long

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Unknown.

PickIconDlg is officially undocumented.

Description & Usage

**PickIconDlg** displays the standard Windows "icon selection" dialog box. It allows the user to choose an icon found inside a file. The function then reports which icon the user selected, if any.

**Windows NT, 2000:** All strings used by this function must be Unicode. Therefore, any strings passed to the function must first be converted into Unicode. Likewise, any strings output by the function also must be converted from Unicode into ANSI.

Return Value

If successful, the function returns a non-zero value. If an error occurred or the user did not choose an icon (for example, the user clicked "Cancel"), the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

**hwndOwner**

A handle to the window which is opening the icon selection dialog box.

**lpstrFile**

A null-terminated string specifying the default icon-containing file to look inside. This string will also receive the name of the file which holds the user’s selection. Therefore, this string must have enough empty space after the terminating null character to receive the final string.
nMaxFile
The length of the string passed as lpstrFile.

lpdwIconIndex
The zero-based index of the icon to select by default. This variable also receives the zero-based index of the icon the user selected.

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function PickIconDlg Lib "shell32.dll" Alias "#62" (ByVal hwndOwner As Long, _
    ByVal lpstrFile As String, ByVal nMaxFile As Long, lpdwIconIndex As Long) As Long
Public Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type
Public Const VER_PLATFORM_WIN32_NT = 2
Public Const VER_PLATFORM_WIN32_WINDOWS = 1
Public Declare Function GetVersionEx Lib "kernel32.dll" Alias "GetVersionExA" _
    (lpVersionInformation As OSVERSIONINFO) As Long
Public Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst _
    As Long, ByVal lpszExeFileName As String, ByVal nIconIndex As Long) As Long
Public Declare Function DrawIcon Lib "user32.dll" (ByVal hDC As Long, ByVal x As Long, _
    ByVal y As Long, ByVal hIcon As Long) As Long
Public Declare Function DestroyIcon Lib "user32.dll" (ByVal hIcon As Long) As Long
    (ByVal lpBuffer As String, ByVal nSize As Long) As Long

' When the user presses button Command1, display an icon selection dialog box. If the user chooses an icon, then draw it in the corner of window Form1's client area.
Private Sub Command1_Click()
    Dim iconfile As String  ' file that contains the desired icon
    Dim iconindex As Long  ' index of the desired icon
    Dim slength As Long  ' length of returned string
    Dim hIcon As Long  ' handle to the icon once it is extracted
    Dim ovi As OSVERSIONINFO  ' identifies the Windows platform
    Dim retval As Long  ' return value

    ' First, determine if the computer is running Windows NT or 2000. In either
case,
    All strings used with PickIconDlg must be converted into Unicode.
ovi.dwOSVersionInfoSize = Len(ovi)
retval = GetVersionEx(ovi)

    Figure out where the System directory is.
iconfile = Space(256)
slength = GetSystemDirectory(iconfile, Len(iconfile))
iconfile = Left(iconfile, slength)

    Have the default selection be the third icon in pifmgr.dll. Include plenty of extra space in the string so it can receive the user's selection.
iconfile = iconfile & "\pifmgr.dll" & vbNullChar & Space(256)
iconindex = 2

    Display the icon selection dialog. If Windows NT or 2000 is running, convert the string to and from Unicode immediately before and after calling the function.
If ovi.dwPlatformId = VER_PLATFORM_WIN32_NT Then
    iconfile = StrConv(iconfile, vbUnicode)
End If
retval = PickIconDlg(Form1.hWnd, iconfile, Len(iconfile), iconindex)
If ovi.dwPlatformId = VER_PLATFORM_WIN32_NT Then
    iconfile = StrConv(iconfile, vbFromUnicode)
End If
' Remove the terminating null and empty space from the string.
iconfile = Left(iconfile, InStr(iconfile, vbNullChar) - 1)

    If the user selected something, draw the icon on Form1.
If retval <> 0 Then
    ' Extract the icon from the file.
hIcon = ExtractIcon(App.hInstance, iconfile, iconindex)
    ' Draw it in the corner of the window.
retval = DrawIcon(Form1.hDC, 0, 0, hIcon)
    ' Destroy the icon to free resources.
retval = DestroyIcon(hIcon)
End If
End Sub

Category

Shell

Back to the Function list.
Back to the Reference section.

Last Modified: June 4, 2000
Pie Function

Declare Function Pie Lib "gdi32.dll" (ByVal hdc As Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long, ByVal X3 As Long, ByVal Y3 As Long, ByVal X4 As Long, ByVal Y4 As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

Pie draws an elliptical pie wedge on a device. The pie wedge is drawn using the device's currently selected pen and is filled using its currently selected brush. The pie wedge consists of two radials from the ellipse's center to the ellipse's edge, filling the area between them (going counterclockwise around the ellipse). The first two sets of (x,y) coordinate pairs specify the bounding rectangle which determines the ellipse. The last two sets of (x,y) pairs determine the points along the ellipse; the start and endpoints are determined by the intersection of a ray from the ellipse's center through the (x,y) coordinate and the ellipse. The function returns 1 if successful, or 0 if an error occurred.

**hdch**
A device context to the device to draw the chord on.

*X1*
The x-coordinate of the upper-left corner of the ellipse's bounding rectangle.

*Y1*
The y-coordinate of the upper-left corner of the ellipse's bounding rectangle.

*X2*
The x-coordinate of the lower-right corner of the ellipse's bounding rectangle.

*Y2*
The y-coordinate of the lower-right corner of the ellipse's bounding rectangle.

*X3*
The x-coordinate of the point determining the starting point of the pie wedge.

*Y3*
The y-coordinate of the point determining the starting point of the pie wedge.

*X4*
The x-coordinate of the point determining the ending point of the pie wedge.

*Y4*
The y-coordinate of the point determining the ending point of the pie wedge.

Example:

' Draw a pie wedge on window Form1. The ellipse has a bounding rectangle ' of (10,20)-(210,120). The pie wedge will have endpoints on the ellipse of (210,70) ' and (110,20) -- i.e., the "upper-right" fourth of the ellipse. Draw the pie wedge ' using Form1's current brush and pen.
Dim retval As Long ' return value

' Draw the chord as specified above.
retval = Pie(Form1.hDC, 10, 20, 210, 120, 210, 70, 110, 20)
See Also: Chord, Ellipse
Category: Filled Shapes

Go back to the alphabetical Function listing.
Go back to the Reference section index.
PlaySound Function

Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (ByVal lpszName As String, ByVal hModule As Long, ByVal dwFlags As Long) As Long

Alternate Declare for when using a resource or memory location:
Declare Function PlaySound_Res Lib "winmm.dll" Alias "PlaySoundA" (ByVal lpszName As Long, ByVal hModule As Long, ByVal dwFlags As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PlaySound plays a waveform sound through the speakers. This sound could be a .wav file, a system event sound (such as the system startup sound), or a sound resource stored in an application. Note that when the function needs to play an application resource or a RAM-loaded sound, Visual Basic users must use the alternate declare of the function in order to pass the numeric identifier of the sound instead of a string. The function returns 0 if an error occurred, or a non-zero value if successful.

*lpszName*
The name or some other identifier of the sound. Its exact format depends on the flags passed as dwFlags.

*hModule*
A handle to the application module containing the sound resource the play, if needed. If the function does not need this information, pass 0 for this parameter.

*dwFlags*
Zero or more of the following flags specifying what lpszName refers to and how to play the sound:

SND_ALIAS = &H10000
lpszName is a string identifying the name of the system event sound to play.

SND_ALIAS_ID = &H110000
lpszName is a string identifying the name of the predefined sound identifier to play.

SND_APPLICATION = &H80
lpszName is a string identifying the application-specific event association sound to play.

SND_ASYNC = &H1
Play the sound asynchronously -- return immediately after beginning to play the sound and have it play in the background.

SND_FILENAME = &H20000
lpszName is a string identifying the filename of the .wav file to play.

SND_LOOP = &H8
Continue looping the sound until this function is called again ordering the looped playback to stop. SND_ASYNC must also be specified.

SND_MEMORY = &H4
lpszName is a numeric pointer refering to the memory address of the image of the waveform sound loaded into RAM.

SND_NODEFAULT = &H2
If the specified sound cannot be found, terminate the function with failure instead of playing the SystemDefault sound. If this flag is not specified, the SystemDefault sound will play if the specified sound cannot be located.
and the function will return with success.

SND_NOSTOP = &H10
If a sound is already playing, do not prematurely stop that sound from playing and instead return with failure. If this flag is not specified, the playing sound will be terminated and the sound specified by the function will play instead.

SND_NOWAIT = &H2000
If a sound is already playing, do not wait for the currently playing sound to stop and instead return with failure.

SND_PURGE = &H40
Stop playback of any waveform sound. lpszName must be an empty string.

SND_RESOURCE = &H4004
lpszName is the numeric resource identifier of the sound stored in an application. hModule must be specified as that application's module handle.

SND_SYNC = &H0
Play the sound synchronously -- do not return until the sound has finished playing.

Example:

' First play the SystemStart event sound synchronously. Then loop
' playing the file C:\Sounds\scream.wav for 5 seconds before stopping.
Dim retval As Long ' return value of the function

' Synchronously play the SystemStart sound. This function returns when the sound is done.
retval = PlaySound("SystemStart", 0, SND_ALIAS Or SND_SYNC)

' Now loop the .wav file for five seconds before purging its playback. Note that
' we don't want the default sound to play if the file is not found.
retval = PlaySound("C:\Sounds\scream.wav", 0, SND_FILENAME Or SND_ASYNC Or
SND_NODEFAULT Or SND_LOOP)
Sleep 5000 ' wait for 5 seconds while sound loops
retval = PlaySound("", 0, SND_PURGE Or SND_NODEFAULT) ' stop playback

See Also: sndPlaySound

Category: Audio
PolyBezier Function

Declare Function PolyBezier Lib "gdi32.dll" (ByVal hdc As Long, lppt As POINT_TYPE, ByVal cPoints As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

**PolyBezier** draws a series of cubic Bézier curves on a device. Each Bézier curve is drawn using the device's current pen. The function begins drawing the curves at a start point and draws each subsequent curve starting at the point where the previous one ended (for example, if the second curve ended at (100,50), the third curve would begin at (100,50) as well).

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

**hdc**

A handle to a device context to the device to draw the Bézier curves on.
**lppt**

An array containing all the end points and control points for the Bézier curves. The first point is the starting point for the first curve. Each set of three points following the first identifies how to draw each curve: the first two points in the set are the two control points, and the third point is the end point of the curve.

**cPoints**

The number of elements in the array passed as *lppt*. This value will have to be one more than three times the number of curves to draw.

**Example**

```vbscript
' This code is licensed according to the terms and conditions listed here.

' Draw two Bézier curves on window Form1 using the solid black stock pen. The coordinates of the curves appear in the example code.
Dim hPen As Long  ' handle to the solid black stock pen
Dim hOldPen As Long  ' handle to Form1's previous pen
Dim pts(0 To 6) As POINT_TYPE  ' array of points for the curves
Dim retval As Long  ' return value

' Load the seven points defining the two Bézier curves into the array. The first four points define the first curve.
pts(0).x = 100: pts(0).y = 100  ' 1st curve starts at (100,100)
pts(1).x = 125: pts(1).y = 75   ' control point (125,75)
pts(2).x = 150: pts(2).y = 125  ' control point (150,125)
pts(3).x = 175: pts(3).y = 100  ' 1st curve ends at (175,100)
' The next three define the second curve, starting at (175,100)
pts(4).x = 225: pts(4).y = 50   ' control point (225,50)
pts(5).x = 300: pts(5).y = 150  ' control point (300, 150)
pts(6).x = 250: pts(6).y = 200  ' 2nd curve ends at (250,200)

' Get a handle to the solid black stock pen.
hPen = GetStockObject(BLACK_PEN)
' Select the pen for use in Form1.
hOldPen = SelectObject(Form1.hDC, hPen)
' Draw the two Bézier curves.
retval = PolyBezier(Form1.hDC, pts(0), 7)
' Select the previous pen used by Form1.
retval = SelectObject(Form1.hDC, hOldPen)
```

**See Also**
PolyBezierTo Function

Declare Function PolyBezierTo Lib "gdi32.dll" (ByVal hdc As Long, lppt As POINT_TYPE, ByVal cCount As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

PolyBezierTo draws a series of cubic Bézier curves on a device. Each Bézier curve is drawn using the device's current pen. The function begins drawing the curves at the device's current point and draws each subsequent curve starting at the point where the previous one ended (for example, if the second curve ended at (100,50), the third curve would begin at (100,50) as well).

Return Value

If an error occurred, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

(hdc)

A handle to a device context to the device to draw the Bézier curves on.
**lppt**

An array containing all the end points and control points for the Bézier curves. Each set of three points identifies one curve: the first two points are the control points, and the third point is the end point of the curve.

**cPoints**

The number of elements in the array passed as lppt. This value will have to be three times the number of curves to draw.

### Example

' This code is licensed according to the terms and conditions listed here.

' Use the solid black stock pen to draw a straight line,
' followed by two Bézier curves on window Form1. The
' coordinates of the line and curves appear in the example code.
Dim hPen As Long  ' handle to the solid black stock pen
Dim hOldPen As Long  ' handle to Form1's previous pen
Dim curpt As POINT_TYPE  ' receives previous current point
Dim pts(0 To 5) As POINT_TYPE  ' array of points for the curves
Dim retval As Long  ' return value

' Get a handle to the solid black stock pen.
hPen = GetStockObject(BLACK_PEN)
' Select the pen for use in Form1.
hOldPen = SelectObject(Form1.hDC, hPen)

' Move the current point to (200, 25) and draw a line
' to the point (100,100).
retval = MoveToEx(Form1.hDC, 200, 25, curpt)
retval = LineTo(Form1.hDC, 100, 100)

' Load the six points defining the two Bézier curves
' into the array. The first three points define the first curve.
' The curve will of course start of (100,100).
pts(0).x = 125: pts(0).y = 75   ' control point (125,75)
pts(1).x = 150: pts(1).y = 125  ' control point (150,125)
pts(2).x = 175: pts(2).y = 100  ' 1st curve ends at (175,100)
' The next three define the second curve, starting at (175,100)
pts(3).x = 225: pts(3).y = 50   ' control point (225,50)
pts(4).x = 300: pts(4).y = 150  ' control point (300, 150)
pts(5).x = 250: pts(5).y = 200  ' 2nd curve ends at (250,200)

' Draw the two Bézier curves.
retval = PolyBezierTo(Form1.hDC, pts(0), 6)
' Select the previous pen used by Form1.
retval = SelectObject(Form1.hDC, hOldPen)

See Also

PolyBezier

Category

Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 11, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/p/polybezierto.html
Polygon Function

Declare Function Polygon Lib "gdi32.dll" (ByVal hdc As Long, lpPoint As POINT_TYPE, ByVal nCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

Polygon draws and fills a polygon on a device. The polygon is drawn using the current pen and is filled using the current brush. The vertices of the polygon are stored in an array passed as lpPoint, in sequential order. Only specify each point once. The function returns 1 if successful, or 0 if an error occurred.

hdc
A device context to the device to draw the polygon on.

lpPoint
An array holding the vertices of the polygon in the order which they will be drawn in. Specify each point only once.

nCount
The number of elements in the array passed as lpPoint.

Example:

' Draw a diamond on window Form1 using the default pen and
' brush. Note how the points must be loaded into the array before calling the
' function.
Dim points(0 To 3) As POINT_TYPE  ' vertices of the polygon
Dim retval As Long    ' return value

' Load the coordinates of the diamond's vertices into the array.
points(0).x = 200: points(0).y = 100  ' 1st point (200,100)
points(1).x = 250: points(1).y = 150  ' 2nd point (250,150)
points(2).x = 200: points(2).y = 200  ' 3rd point (200,200)
points(3).x = 150: points(3).y = 150  ' 4th point (150,150)

' Draw the diamond using Form1's default pen and brush
retval = Polygon(Form1.hDC, points(0), 4)  ' four points in the array

See Also: PolyPolygon
Category: Filled Shapes

Go back to the alphabetical Function listing.
Polyline Function

Declare Function Polyline Lib "gdi32.dll" (ByVal hdc As Long, lpPoint As POINT_TYPE, ByVal nCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

Polyline draws a series of lines on a graphics-capable device. These lines connect the points given in an array passed as lpPoint. Lines are drawn connecting the first point to the second point, the second point to the third point, etc. The first and last points are not connected. The lines are drawn using the device's current drawing color. The function returns 1 if successful, or 0 if an error occurred.

hdc
The device context of the device to draw the lines on.

lpPoint
An array specifying the x and y coordinates of each point to draw lines to or from.

nCount
The number of elements in the array passed as lpPoint.

Example:

' Draw a triangle having corners (100,100), (200, 150), and (0, 150)
' on window Form1. Note how since we want the first and last points to be connected,
' point (100,100) must be given as both the first and last points.
Dim points(0 To 3) As POINT_API ' the points to draw to/from
Dim retval As Long ' return value

' Put the points to use into the array. Four points must be specified to draw the
' triangle because the point (100,100) must be entered twice.
points(0).x = 100: points(0).y = 100 ' point #0: (100,100)
points(1).x = 200: points(1).y = 150 ' point #1: (200,150)
points(2).x = 0: points(2).y = 150 ' point #2: (0,150)
points(3).x = 100: points(3).y = 100 ' point #3: (100,100)

Form1.ForeColor = RGB(255, 0, 0) ' set Form1's drawing color to red
retval = Polyline(Form1.hDC, points(0), 4) ' draw the lines

See Also: LineTo, PolylineTo, PolyPolyline
Category: Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.
PolylineTo Function

Declare Function PolylineTo Lib "gdi32.dll" (ByVal hdc As Long, lppt As POINT_TYPE, ByVal cCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PolylineTo draws a series of connected lines on a graphics-capable device. The points are given to the function inside an array passed as lppt. The function draws lines connecting the device’s current point to the first point, the first point to the second point, the second point to the third point, etc. When the function is finished, it sets the current point to whatever the last point in the array is. The original current point and the last point are not connected. The lines are drawn in the device’s current drawing color. The function returns 1 if successful, or 0 if an error occurred.

hdc
The device context of the device to draw the lines on.

lppt
An array of points to connect using the lines.

cCount
The number of elements in lppt.

Example:

' Draw a red triangle with corners (100,100), (200,150), and (0,150)
' on window Form1. The current point must first be set to (100,100), and the last
' point must also be given as (100,100) to close the triangle.
Dim points(0 To 2) As POINT_TYPE  ' points given to the function
Dim curpt As POINT_TYPE  ' receives current point from MoveToEx
Dim retval As Long  ' return value

' Set Form1's current point to (100,100)
retval = MoveToEx(Form1.hDC, 100, 100, curpt)

' Load the points of the triangle into the array points(). Notice that (100,100)
' is given as the last point to close the figure.
points(0).x = 200: points(0).y = 150  ' point #0: (200,150)
points(1).x = 0: points(1).y = 150     ' point #1: (0,150)
points(2).x = 100: points(2).y = 100  ' point #2: (100,100)
Form1.ForeColor = RGB(255, 0, 0)  ' set Form1's drawing color to red
retval = PolylineTo(Form1.hDC, points(0), 3)  ' draw the lines

See Also: LineTo, Polyline, PolyPolyline
PolyPolygon Function

Declare Function PolyPolygon Lib "gdi32.dll" (ByVal hdc As Long, lpPoint As POINT_TYPE, lpPolyCounts As Long, ByVal nCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PolyPolygon draws multiple polygons on a given device. The polygons are drawn using the device's current pen and are filled using the device's current brush. The various polygons are not connected. The vertices of every polygon are stored in an array passed as lpPoint. The array passed as lpPolyCounts identifies the number of vertices in each polygon. This function is equivalent to calling the Polygon function multiple times. The function returns 1 if successful, or 0 if an error occurred.

hdc
A device context to the device to draw the polygons on.

lpPoint
An array holding all of the vertices of every polygon to draw, listed in their proper order. Do not specify any point more than once in each polygon's set of vertices.

lpPolyCounts
An array holding the number of points in lpPoint belonging to each polygon. For example, lpPolyCounts(0) identifies the number of points belonging to the first polygon, etc.

nCount
The number of elements in the array passed as lpPolyCounts.

Example:

' Draw a diamond and a triangle on window Form1. The two
' polygons are not interconnected and are drawn using the window's current pen
' and brush. Note how the two arrays delineate each polygon's set of vertices.
Dim points(0 To 6) As POINT_TYPE  ' holds diamond's and triangle's vertices
Dim numpoints(0 To 1) As Long  ' holds number of points belonging to each polygon
Dim retval As Long  ' return value

' Load the points belonging to the diamond.
points(0).x = 200: points(0).y = 100  ' 1st point: (200,100)
points(1).x = 250: points(1).y = 150  ' 2nd point: (250,150)
points(2).x = 200: points(2).y = 200  ' 3rd point: (200,200)
points(3).x = 150: points(3).y = 150  ' 4th point: (150,150)
numpoints(0) = 4  ' first four points identify the diamond

' Load the points belonging to the triangle.
points(4).x = 350: points(4).y = 200  ' 1st point: (350,200)
points(5).x = 400: points(5).y = 250  ' 2nd point: (400,250)
points(6).x = 300: points(6).y = 250  ' 3rd point: (300,250)
numpoints(1) = 3  ' next three points identify the triangle

' Draw the two polygons
retval = PolyPolygon(Form1.hDC, points(0), numpoints(0), 2)  ' two polygons

See Also: Polygon
Category: Filled Shapes

Go back to the alphabetical Function listing.
Go back to the Reference section index.
PolyPolyline Function

Declare Function PolyPolyline Lib "gdi32.dll" (ByVal hdc As Long, lppt As POINT_TYPE, lpdwPolyPoints As Long, ByVal cCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PolyPolyline draws multiple sets of lines connecting points on a graphics-capable device. This function has a similar effect as using Polyline with various sets of points. Lines are drawn to connect the first point in a set to the second point, the second point to the third, etc. The first and last points in a set are not connected, just as the sets are not connected to each other. All of the points go into an array passed as lppt. The connecting lines are drawn in the object’s current drawing color. The function returns 1 if successful, or 0 if an error occurred.

hc
The device context of the device to draw the lines on.

lppt
An array holding all of the points in every set. This should have all of the points in the first set, followed by those in the second set, followed by those in the third set, etc.

lpdwPolyPoints
An array identifying how many points belong to each set. The first element specified how many points are in the first set, etc.

cCount
The number of elements in lpdwPolyPoints.

Example:

' Draw a red triangle and a red parallelogram on window Form1. The ' triangle has corners (100,100), (200,150), and (0,150). The parallelogram has corners ' (300,300), (400,300), (350,400), and (250,400). Note that since the first and last points ' are not connected, the beginning point is also specified as the last point in order ' to close the figures.
Dim points(0 To 8) As POINT_TYPE ' the points in both sets
Dim numpoints(0 To 1) As Long ' the number of points in each set
Dim retval As Long ' return value

' Load the points for the triangle into the array. The point (100,100) is both the first ' and last points in order to close the figure.
points(0).x = 100: points(0).y = 100 ' point #0: (100,100)
points(1).x = 200: points(1).y = 150 ' point #1: (200,150)
points(2).x = 0: points(2).y = 150 ' point #2: (0,150)
points(3).x = 100: points(3).y = 100 ' point #3: (100,100)
numpoints(0) = 4 ' number of points in first set

' Load the points for the parallelogram in a similar fashion.
points(4).x = 300: points(4).y = 300 ' point #4: (300,300)
points(5).x = 400: points(5).y = 300 ' point #5: (400,300)
points(6).x = 350: points(6).y = 400 ' point #6: (350,400)
points(7).x = 250: points(7).y = 400 ' point #7: (250,400)
points(8).x = 300: points(8).y = 300 ' point #8: (300,300)
numpoints(1) = 5 ' number of points in second set

' Now, finally draw both sets of points. The two are not mutually connected.
Form1.ForeColor = RGB(255, 0, 0) ' set Form1's drawing color to red
retval = PolyPolyline(Form1.hDC, points(0), numpoints(0), 2) ' draw the lines

See Also: LineTo, Polyline, PolylineTo
Category: Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.
PrintDlg Function

Declare Function PrintDlg Lib "comdlg32.dll" Alias "PrintDlgA" (pPrintdlg As PRINTDLG_TYPE) As Long

Platforms: Win 32s, Win 95/98, Win NT

PrintDlg displays either the Print common dialog box or the Print Setup dialog box. Either box can be used to allow the user to select a printer and other settings, such as the number of copies and the page range, desired for a print operation. Information for initializing the dialog box as well as information returned from it is stored in the structure passed as pPrintdlg. See the pages for the PRINTDLG_TYPE, DEVMODE, and DEVNAMES structures for more details about using this function. Note that instead of using the latter two structures explicitly, handles to the memory blocks holding their data are required by the function; see the example below. The function returns 0 if either an error occurred or the user pressed Cancel, or a non-zero value if the user successfully pressed OK.

pPrintdlg
Stores both the dialog box's initialization settings and the information returned from the dialog box.

Example:

' Open a Print common dialog box. Then display certain selections the user made, ' such as the printer name, number of copies, and orientation. Carefully note how memory ' blocks are allocated to hold the two data structures containing information about the ' printer device. To save space, Visual Basic's Printer object, referring to the default ' printer, is used to provide the defaults. Of course, API functions could also be used ' to get these defaults.
Dim pd As PRINTDLG_TYPE ' holds information to make the dialog box
Dim printmode As DEVMODE ' holds settings for the printer device
Dim printnames As DEVNAMES ' holds device, driver, and port names
Dim pMode As Long, pNames As Long ' pointers to the memory blocks for the two DEV* structures
Dim retval As Long ' return value of function

' First, load default settings into printmode. Note that we only fill relevant information.
printmode.dmDeviceName = Printer.DeviceName ' name of the printer
printmode.dmSize = Len(printmode) ' size of the data structure
printmode.dmFields = DM_ORIENTATION ' identify which other members have information
printmode.dmOrientation = DMORIENT_PORTRAIT ' default to Portrait orientation
' Next, load strings for default printer into printnames. Note the unusual way in which such
' information is stored. This is explained on the DEVNAMES page.
devnames.wDriverOffset = 8 ' offset of driver name string
devnames.wDeviceOffset = devnames.wDriverOffset + 1 + Len(Printer.DriverName) ' offset of printer name string
devnames.wOutputOffset = devnames.wDeviceOffset + 1 + Len(Printer.Port) ' offset to output port string
devnames.wDefault = 0 ' maybe this isn't the default selected printer
' Load the three strings into the buffer, separated by null characters.
devnames.extra = Printer.DriverName & vbCrLf & Printer.DeviceName & vbCrLf & Printer.Port & vbCrLf

' Finally, load the initialization settings into pd, which is passed to the function. We'll
' set the pointers to the structures after this.
pd.lStructSize = Len(pd) ' size of structure
dhwndOwner = Form1.hWnd ' window Form1 is opening the Print dialog box
' Flags: All Pages default, disable Print to File option, return device context:
pd.flags = PD_ALLPAGES Or PD_DISABLEPRINTTOFILE Or PD_RETURNDC
pd.nMinPage = 1 ' allow user to select first page of "document"
pd.nMaxPage = 15 ' let's say there are 15 pages of the "document"
' Note how we can ignore those members which will be set or are not used here.
' Copy the data in printmode and printnames into the memory blocks we allocate.
pd.hDevMode = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(printmode)) ' allocate memory block
pMode = GlobalLock(pd.hDevMode) ' get a pointer to the block
CopyMemory ByVal pMode, printmode, Len(printmode) ' copy structure to memory block
retval = GlobalUnlock(pd.hDevMode) ' unlock the block
' Now do the same for printnames.
pd.hDevNames = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(printnames)) ' allocate memory block
pNames = GlobalLock(pd.hDevNames) ' get a pointer to the block
CopyMemory ByVal pNames, printnames, Len(printnames) ' copy structure to memory block
retval = GlobalUnlock(pd.hDevNames) ' unlock the block

' Finally, open the dialog box!
retval = PrintDlg(pd) ' looks so simple, doesn't it?

' If the user hit OK, display some information about the selection.
If retval <> 0 Then
' First, we must copy the memory block data back into the structures. This is almost identical
' to the code above where we did the reverse. Comments here are omitted for brevity.
pMode = GlobalLock(pd.hDevMode)
CopyMemory printmode, ByVal pMode, Len(printmode)
retval = GlobalUnlock(pd.hDevMode)
pNames = GlobalLock(pd.hDevNames)
CopyMemory printnames, ByVal pNames, Len(printnames)

retval = GlobalUnlock(pd.hDevNames)

' Now, display the information we want.  We could instead use this info to print something.
Debug.Print "Printer Name: "; printmode.dmDeviceName
Debug.Print "Number of Copies: "; pd.nCopies
Debug.Print "Orientation: ";
If printmode.dmOrientation = DMORIENT_PORTRAIT Then Debug.Print "Portrait" Else Debug.Print "Landscape"
If (pd.flags And PD_SELECTION) = PD_SELECTION Then ' user chose "Selection"
  Debug.Print "Print the current selection."
Elseif (pd.flags And PD_PAGENUMS) = PD_PAGENUMS Then ' user chose a page range
  Debug.Print "Print pages "; pd.nFromPage; " to "; pd.nToPage
Else ' only one left is "All"
  Debug.Print "Print all pages."
End If
End If

' No matter what, we have to deallocate the memory blocks from before.
retval = GlobalFree(pd.hDevMode)
retval = GlobalFree(pd.hDevNames)

Category: Common Dialog

Go back to the alphabetical Function listing.
Go back to the Reference section index.
PrinterProperties Function

Declare Function PrinterProperties Lib "winspool.drv" (ByVal hWnd As Long, ByVal hPrinter As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

PrinterProperties displays the Properties dialog box for a specific printer. This dialog allows the user to configure the printer, and its look generally depends on the make and model of printer. The dialog is opened modally -- your program pauses execution until the user exits the dialog.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to the window opening the dialog box.

hPrinter
A handle to the printer to display the Properties dialog box for.

Example

Display the Properties dialog box for the system's default printer. The dialog box is opened when the user clicks the button cmdProperties. To use the example, first place a command button named cmdProperties on a form window.
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/p/printerproperties.html).

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

```vbnet
Public Type PRINTER_INFO_1
  flags As Long
  pDescription As String
  pName As String
  pComment As String
End Type

Public Declare Function EnumPrinters Lib "winspool.drv" Alias "EnumPrintersA" (ByVal flags As Long, ByVal name As String, ByVal Level As Long, pPrinterEnum As Long, ByVal cdBuf As Long, ByVal pcbNeeded As Long, pcReturned As Long) As Long
Public Const PRINTER_ENUM_DEFAULT = &H1
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function OpenPrinter Lib "winspool.drv" Alias "OpenPrinterA" (ByVal pPrinterName As String, phPrinter As Long, pDefault As Any) As Long
Public Declare Function PrinterProperties Lib "winspool.drv" (ByVal hWnd As Long, ByVal hPrinter As Long) As Long
Public Declare Function ClosePrinter Lib "winspool.drv" (ByVal hPrinter As Long) As Long
```

*** Place the following code inside a form window. ***

```vbnet
Private Sub cmdProperties_Click()
  Dim pi1 As PRINTER_INFO_1 ' a little info about the printer
  Dim bytesNeeded As Long ' size needed for buffer
  Dim numPrinters As Long ' number of printers enumerated (should be 1)
  Dim buffer() As Long ' buffer for printer information
  Dim slength As Long ' length of string to copy
  Dim hPrinter As Long ' handle to the printer
  Dim retval As Long ' generic return value

  ' Figure out how much space is needed to store the printer information.
  retval = EnumPrinters(PRINTER_ENUM_DEFAULT, vbNullString, 1, ByVal 0, 0, bytesNeeded, numPrinters)
  ' Allocate that much space in the buffer array.
  ReDim buffer(0 To bytesNeeded / 4 - 1) As Long
  ' Get information about the default printer.
  retval = EnumPrinters(PRINTER_ENUM_DEFAULT, vbNullString, 1, buffer(0), bytesNeeded, _
                         bytesNeeded, numPrinters)
  ' Make sure we were successful.
```
If retval = 0 Or numPrinters <> 1 Then
    Debug.Print "No default printer or some other error."
    Exit Sub
End If

' Copy the data into the structure.
With pi1
    ' Copy numerical data directly.
    .flags = buffer(0)
    ' Strings require more work, since the buffer holds pointers to them.
    .pDescription = Space(lstrlen(buffer(1)))
    retval = lstrcpy(.pDescription, buffer(1))
    .pName = Space(lstrlen(buffer(2)))
    retval = lstrcpy(.pName, buffer(2))
    .pComment = Space(lstrlen(buffer(3)))
    retval = lstrcpy(.pComment, buffer(3))
End With

' Open the printer.
retval = OpenPrinter(pi1.pName, hPrinter, ByVal CLng(0))
If retval <> 0 Then
    ' Display the properties dialog.
    retval = PrinterProperties(Me.hWnd, hPrinter)
    ' Close the printer.
    retval = ClosePrinter(hPrinter)
Else
    Debug.Print "Unable to open printer!"
End If

End Sub

Category

Printers

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/p/printerproperties.html
Process32First Function

Declare Function Process32First Lib "kernel32.dll" (ByVal hSnapshot As Long, lppe As PROCESSENTRY32) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Description & Usage

Process32First retrieves information about the first process in the process list contained in a system snapshot. This snapshot must have previously been generated using the CreateToolhelp32Snapshot function.

Return Value

If successful, the function returns a nonzero value. If an error occurred, possibly if there is an empty or nonexistent process list in the snapshot, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

hSnapshot

A handle to the system snapshot created by the CreateToolhelp32Snapshot function. This snapshot contains the process list from which the first process is read.

lppe

Receives information about the first process in the snapshot. The dwSize member of this structure must be initialized before calling the function.

Example

Print a list of all the processes currently running on the computer when the user clicks button Command1. To do this, a
snapshot of the running process list is taken, and then each process in it is analyzed. The filename of the process and the
number of threads owned by it is then displayed. To use this example, place a command button named Command1 on a form
window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreateToolhelp32Snapshot Lib "kernel32.dll" (ByVal dwFlags As Long, ByVal th32ProcessID As Long) As Long
Public Const TH32CS_SNAPPROCESS = &H2
Public Type PROCESSENTRY32
dwSize As Long
cntUsage As Long
th32ProcessID As Long
th32DefaultHeapID As Long
th32ModuleID As Long
cntThreads As Long
th32ParentProcessID As Long
pcPriClassBase As Long
dwFlags As Long
szExeFile As String * 260
End Type
Public Declare Function Process32First Lib "kernel32.dll" (ByVal hSnapshot As Long, ByVal lppe As PROCESSENTRY32) As Long
Public Declare Function Process32Next Lib "kernel32.dll" (ByVal hSnapshot As Long, ByVal lppe As PROCESSENTRY32) As Long
Public Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim hSnapshot As Long  ' handle to the snapshot of the process list
    Dim processInfo As PROCESSENTRY32  ' information about a process in that list
    Dim success As Long  ' success of having gotten info on another process
    Dim exeName As String  ' filename of the process
    Dim retval As Long  ' generic return value

    ' First, make a snapshot of the current process list.
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)

    ' Get information about the first process in the list.
    processInfo.dwSize = Len(processInfo)
    success = Process32First(hSnapshot, processInfo)
    ' Make sure a handle was returned.
    If hSnapshot = -1 Then
        Debug.Print "Unable to take snapshot of process list!"
        Exit Sub
    End If

    ' This code is licensed according to the terms and conditions listed here.

    ' Declarations and such needed for the example:
    ' (Copy them to the (declarations) section of a module.)
    Public Declare Function CreateToolhelp32Snapshot Lib "kernel32.dll" (ByVal dwFlags As Long, ByVal th32ProcessID As Long) As Long
    Public Const TH32CS_SNAPPROCESS = &H2
    Public Type PROCESSENTRY32
dwSize As Long
cntUsage As Long
th32ProcessID As Long
th32DefaultHeapID As Long
th32ModuleID As Long
cntThreads As Long
th32ParentProcessID As Long
pcPriClassBase As Long
dwFlags As Long
szExeFile As String * 260
End Type
Public Declare Function Process32First Lib "kernel32.dll" (ByVal hSnapshot As Long, ByVal lppe As PROCESSENTRY32) As Long
Public Declare Function Process32Next Lib "kernel32.dll" (ByVal hSnapshot As Long, ByVal lppe As PROCESSENTRY32) As Long
Public Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim hSnapshot As Long  ' handle to the snapshot of the process list
    Dim processInfo As PROCESSENTRY32  ' information about a process in that list
    Dim success As Long  ' success of having gotten info on another process
    Dim exeName As String  ' filename of the process
    Dim retval As Long  ' generic return value

    ' First, make a snapshot of the current process list.
    hSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0)

    ' Get information about the first process in the list.
    processInfo.dwSize = Len(processInfo)
    success = Process32First(hSnapshot, processInfo)
    ' Make sure a handle was returned.
    If hSnapshot = -1 Then
        Debug.Print "Unable to take snapshot of process list!"
        Exit Sub
    End If

End Sub
Loop for each process on the list.
While success <> 0
    ' Extract the filename of the process (i.e., remove the empty space)
    exeName = Left(processInfo.szExeFile, InStr(processInfo.szExeFile, vbNullChar) - 1)
    ' Display the process name and the number of threads it owns.
    Debug.Print "Process: "; exeName
    Debug.Print " - Number of threads:"; processInfo.cntThreads
    ' Get information about the next process, if there is one.
    processInfo.dwSize = Len(processInfo)
    success = Process32Next(hSnapshot, processInfo)
Wend

' Destroy the snapshot, now that we no longer need it.
retval = CloseHandle(hSnapshot)
End Sub

See Also

Process32Next

Category

Tool Help

Back to the Function list.
Back to the Reference section.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/p/process32first.html
Process32First Function

Declare Function Process32Next Lib "kernel32.dll" (ByVal hSnapshot As Long, lppe As PROCESSENTRY32) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Description & Usage

Process32Next retrieves information about the next unread process in the process list contained in a system snapshot. This snapshot must have previously been generated using the CreateToolhelp32Snapshot function. Calling this function repeatedly, after an initial call to Process32First, will allow your program to read the entire process list.

Return Value

If successful, the function returns a nonzero value. If an error occurred, most likely if there are no more unread processes in the list, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

**hSnapshot**

A handle to the system snapshot created by the CreateToolhelp32Snapshot function. This snapshot contains the process list from which the next unread process is read.

**lppe**

Receives information about the next unread process in the snapshot. The dwSize member of this structure must be initialized before calling the function.

Example
Print a list of all the processes currently running on the computer when the user clicks button Command1. To do this, a snapshot of the running process list is taken, and then each process in it is analyzed. The filename of the process and the number of threads owned by it is then displayed. To use this example, place a command button named Command1 on a form window.

 This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function CreateToolhelp32Snapshot Lib "kernel32.dll" (ByVal dwFlags As Long, ByVal th32ProcessID As Long) As Long
Public Const TH32CS_SNAPPROCESS = &H2
Public Type PROCESSENTRY32
dwSize As Long
cntUsage As Long
th32ProcessID As Long
th32DefaultHeapID As Long
th32ModuleID As Long
cntThreads As Long
th32ParentProcessID As Long
pcPriClassBase As Long
dwFlags As Long
szExeFile As String * 260
End Type
Public Declare Function Process32First Lib "kernel32.dll" (ByVal hSnapshot As Long, lppe As PROCESSENTRY32) As Long
Public Declare Function Process32Next Lib "kernel32.dll" (ByVal hSnapshot As Long, lppe As PROCESSENTRY32) As Long
Public Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

*** Place the following code inside a form window. ***

Private Sub Command1_Click()
Dim hSnapshot As Long ' handle to the snapshot of the process list
Dim processInfo As PROCESSENTRY32 ' information about a process in that list
Dim success As Long ' success of having gotten info on another process
Dim exeName As String ' filename of the process
Dim retval As Long ' generic return value

' First, make a snapshot of the current process list.
hSnapshot = CreateToolhelp32Snapshot (TH32CS_SNAPPROCESS, 0)

' Get information about the first process in the list.
processInfo.dwSize = Len(processInfo)
success = Process32First (hSnapshot, processInfo)
' Make sure a handle was returned.
If hSnapshot = -1 Then
    Debug.Print "Unable to take snapshot of process list!"
    Exit Sub
End If
' Loop for each process on the list.
While success <> 0
    ' Extract the filename of the process (i.e., remove the empty space)
    exeName = Left(processInfo.szExeFile, InStr(processInfo.szExeFile, vbNullChar) - 1)
    ' Display the process name and the number of threads it owns.
    Debug.Print "Process: "; exeName
    Debug.Print "  - Number of threads: "; processInfo.cntThreads
    ' Get information about the next process, if there is one.
    processInfo.dwSize = Len(processInfo)
    success = Process32Next(hSnapshot, processInfo)
Wend

' Destroy the snapshot, now that we no longer need it.
retval = CloseHandle(hSnapshot)
End Sub

See Also

Process32First

Category

Tool Help

Back to the Function list.
Back to the Reference section.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/p/process32next.html
PtInRect Function

Declare Function PtInRect Lib "user32.dll" (lpRect As RECT, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PtInRect determines if a point lies inside or outside of a rectangle. Note that Windows considers the left and top edges of a rectangle to be inside it, and the right and bottom edges to be outside. The function returns 1 if the point is inside or 0 if it is outside.

lpRect
The rectangle to look inside.

x
The x coordinate of the point to determine if it is inside or outside.

y
The y coordinate of the point to determine if it is inside or outside.

Example:

' Determine if the mouse cursor is inside or outside of window Form1.
' This is done by checking the point of the mouse cursor with the rectangle of the window.
Dim mousept As POINT_TYPE  ' receives mouse coordinate
Dim winrect As RECT  ' receives rectangle of Form1
Dim isinside As Long  ' receives 1 if inside or 0 if outside
Dim retval As Long  ' return value for other functions

retval = GetCursorPos(mousept)  ' determine the mouse cursor's position
retval = GetWindowRect(Form1.hWnd, winrect)  ' determine Form1's rectangle

' Check to see if the mouse cursor is located inside of the Form1 rectangle
isinside = PtInRect(winrect, mousept.x, mousept.y)
If isinside = 1 Then
    Debug.Print "The mouse cursor is currently inside of Form1."
Else
    Debug.Print "The mouse cursor is currently outside of Form1."
End If

Category: Rectangles

Go back to the alphabetical Function listing.
PtInRegion Function

Declare Function PtInRegion Lib "gdi32.dll" (ByVal hRgn As Long, ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

PtInRegion determines if a given point lies within a region. The point is considered to be inside the region if it is within the area bounded by the region. The function returns 0 if the point is not inside the region, or a non-zero value if the point is in the region.

hRgn
A handle to the region to determine if a given point lies within.

x
The x-coordinate of the point to determine if it lies within the region.

y
The y-coordinate of the point to determine if it lies within the region.

Example:

' Consider a line connecting the upper-right and lower-left corners of the screen, and consider the region made of the upper-left side of this line. Determine
' if the mouse cursor lies within this region or not.
Dim swidth As Long, sheight As Long ' width and height of the screen
Dim hRgn As Long ' handle to the triangular region explained above
Dim curpos As POINT_TYPE ' receives location of mouse cursor
Dim vertices(0 To 2) As POINT_TYPE ' vertices of region to create
Dim isinside As Long ' receives 0 if not inside, non-zero otherwise
Dim retval As Long ' generic return value

' Get the screen's width and height. Use this information to create the region.
swidth = GetSystemMetrics(SM_CXSCREEN) ' screen width
sheight = GetSystemMetrics(SM_CYSCREEN) ' screen height

' Load region's vertices into the array and create it.
vertices(0).x = 0: vertices(0).y = 0 ' vertex #1: upper-left corner of screen
vertices(1).x = swidth: vertices(1).y = 0 ' vertex #2: upper-right corner of screen
vertices(2).x = 0: vertices(2).y = sheight ' vertex #3: lower-left corner of screen
hRgn = CreatePolygonRgn(vertices(0), 3, ALTERNATE) ' create the region

' Get the current position of the mouse cursor.
retval = GetCursorPos(curpos)

' Determine if the cursor location lies within the region.
isinside = PtInRegion(hRgn, curpos.x, curpos.y) ' is the point in the region?
If isinside = 0 Then  ' not inside
    Debug.Print "The cursor is not inside the region."
Else
    Debug.Print "The cursor is inside the region."
End If

' Delete the region to free up resources.
retval = DeleteObject (hRgn)

See Also: RectInRegion
Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
QueryPerformanceCounter Function

Declare Function QueryPerformanceCounter Lib "kernel32.dll" (lpPerformanceCount As LARGE_INTEGER) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Required Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

QueryPerformanceCounter obtains the current reading of the system's high-performance timer. The high-performance timer is a counter that the system increments many times a second. To find out how quickly the timer is incremented, use QueryPerformanceFrequency. If you know its frequency, you can use the high-performance timer to measure small intervals of time precisely. However, keep in mind that not all computers support a high-performance timer.

Return Value

If the system contains a high-performance timer, the function returns a non-zero value. If the system does not contain such a timer, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

lpPerformanceCount

Receives the current 64-bit value of the computer's high-performance timer, if one exists.

Example

Use the high-performance timer to determine how long it takes to calculate the square roots of the first 100,000 positive integers. Run this test when the user clicks on button Command1. Obviously, to use this example, you need to place a
command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type LARGE_INTEGER
    LowPart As Long
    HighPart As Long
End Type
Public Declare Function QueryPerformanceFrequency Lib "kernel32.dll" (lpFrequency As LARGE_INTEGER) As Long
Public Declare Function QueryPerformanceCounter Lib "kernel32.dll" (lpPerformanceCount As LARGE_INTEGER) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

' This function converts the data in a LARGE_INTEGER structure into
' the form of VB's 64-bit Currency data type. For more information about how this works,
' go to http://www.vbapi.com/articles/64bit/index.html.
Public Function LI2Curr (li As LARGE_INTEGER) As Currency
    Dim temp As Currency
    CopyMemory temp, li, 8
    LI2Curr = temp * 10000
End Function

' *** Place the following code inside the form. ***
Private Sub Command1_Click()
    Dim freq As Currency ' high-performance timer frequency
    Dim count1 As Currency ' timer reading before calculation
    Dim count2 As Currency ' timer reading after calculation
    Dim buffer1 As LARGE_INTEGER ' data input buffer for...
    Dim buffer2 As LARGE_INTEGER ' ...timer functions
    Dim c As Long ' counter variable
    Dim result As Double ' result of square root calculation
    Dim retval As Long ' generic return value

    ' Get the frequency of the high-performance timer and, in the process,
    ' determine if it actually exists or not.
    retval = QueryPerformanceFrequency(buffer1)
    If retval = 0 Then
        Debug.Print "This computer does not have a high-performance timer."
        Exit Sub
    End If
    freq = LI2Curr(buffer1)

    ' Time how long it takes to calculate the first 100,000 positive integer
    ' square roots. Do this by comparing the high-performance timer reading
before
    ' and after the series of calculations.
    retval = QueryPerformanceCounter(buffer1)
    For c = 1 To 100000
        result = Sqr(c)
    Next c
    retval = QueryPerformanceCounter(buffer2)

    ' Calculate the time interval between measurements.
    count1 = LI2Curr(buffer1)
    count2 = LI2Curr(buffer2)
    Debug.Print "The calculation took"; (count2 - count1) / freq; "seconds."
End Sub

See Also

QueryPerformanceFrequency

Category

Timers

Back to the Function list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/q/queryperformancecounter.html
QueryPerformanceFrequency Function

Declare Function QueryPerformanceFrequency Lib "kernel32.dll" (lpFrequency As LARGE_INTEGER) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Required Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

QueryPerformanceFrequency obtains the frequency of the system's high-performance timer. This value indicates the number of times the high-performance timer increments itself every second. Knowing this frequency, you can use QueryPerformanceCounter to measure precise time intervals. However, not all systems support a high-performance timer.

Return Value

If the system contains a high-performance timer, the function returns a non-zero value. If the system does not contain such a timer, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

lpFrequency

Receives the 64-bit frequency of the computer's high-performance timer, if one exists.

Example

Use the high-performance timer to determine how long it takes to calculate the square roots of the first 100,000 positive integers. Run this test when the user clicks on button Command1. Obviously, to use this example, you need to place a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Type LARGE_INTEGER
    LowPart As Long
    HighPart As Long
End Type
Public Declare Function QueryPerformanceFrequency Lib "kernel32.dll" (lpFrequency As LARGE_INTEGER) As Long
Public Declare Function QueryPerformanceCounter Lib "kernel32.dll" (lpPerformanceCount As LARGE_INTEGER) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

This function converts the data in a LARGE_INTEGER structure into the form of VB's 64-bit Currency data type. For more information about how this works, go to http://www.vbapi.com/articles/64bit/index.html.

Public Function LI2Curr (li As LARGE_INTEGER) As Currency
    Dim temp As Currency
    CopyMemory temp, li, 8
    LI2Curr = temp * 10000
End Function

*** Place the following code inside the form. ***

Private Sub Command1_Click()
    Dim freq As Currency ' high-performance timer frequency
    Dim count1 As Currency ' timer reading before calculation
    Dim count2 As Currency ' timer reading after calculation
    Dim buffer1 As LARGE_INTEGER ' data input buffer for...
    Dim buffer2 As LARGE_INTEGER ' ...timer functions
    Dim c As Long ' counter variable
    Dim result As Double ' result of square root calculation
    Dim retval As Long ' generic return value

    ' Get the frequency of the high-performance timer and, in the process, determine if it actually exists or not.
    retval = QueryPerformanceFrequency(buffer1)
    If retval = 0 Then
        Debug.Print "This computer does not have a high-performance timer."
        Exit Sub
    End If
    freq = LI2Curr(buffer1)

    ' Time how long it takes to calculate the first 100,000 positive integer square roots. Do this by comparing the high-performance timer reading before... and after the series of calculations.
val = QueryPerformanceCounter (buffer1)
For c = 1 To 100000
    result = Sqr(c)
Next c
val = QueryPerformanceCounter (buffer2)

' Calculate the time interval between measurements.
count1 = LI2Curr(buffer1)
count2 = LI2Curr(buffer2)
Debug.Print "The calculation took"; (count2 - count1) / freq; "seconds."
End Sub

See Also

QueryPerformanceCounter

Category

Timers

Back to the Function list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/q/queryperformancefrequency.html
ReadFile Function

Declare Function ReadFile Lib "kernel32.dll" (ByVal hFile As Long, lpBuffer As Any, ByVal nNumberOfBytesToRead As Long, ByVal lpNumberOfBytesRead As Long, ByVal lpOverlapped As Any) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

ReadFile reads the desired amount of bytes from a file. The file must of course have been opened with at least read-level access. If the file is synchronous (not overlapped), the function begins reading the file from the current position of the file pointer, and the function automatically adjusts the file pointer to point to the byte immediately after the last byte read. If the file is asynchronous (overlapped), the structure passed as lpOverlapped identifies the point to begin reading from, and the program calling the function is responsible for updating the file pointer.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful (including if the function attempted to read past the end of the file), the function returns a non-zero value.

Visual Basic-Specific Issues

When passing a string as lpBuffer to receive the data read from the file, the ByVal keyword must precede the string. The keyword is not necessary for any other data types passed for lpBuffer. When passing 0 for lpOverlapped, the expression ByVal CLng(0) must be used.
**hFile**
A handle to the file to read from. The file must have at least read-level access.

**lpBuffer**
The variable, array, or structure which receives the data read from the file.

**nNumberOfBytesToRead**
The number of bytes of data to read from the file and put into lpBuffer.

**lpNumberOfBytesRead**
Receives the number of bytes of data actually read from the file. If this is less than lpNumberOfBytesToRead, the function has attempted to read beyond the end of the file.

**lpOverlapped**
If the file is asynchronous (overlapped), this is an OVERLAPPED structure specifying where to begin reading from. If the file is synchronous (not overlapped), this must be 0.

## Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/r/readfile.html)

' Read both a Long (32-bit) number and a String from the file C:\Test\myfile.txt. Notice how the ByVal keyword must be used when reading a string variable.
Dim longbuffer As Long  ' receives long read from file
Dim stringbuffer As String  ' receives string read from file
Dim numread As Long  ' receives number of bytes read from file
Dim hFile As Long  ' handle of the open file
Dim retval As Long  ' return value

' Open the file for read-level access.
hFile = CreateFile("C:\Test\myfile.txt", GENERAL_READ, FILE_SHARE_READ, ByVal CLng(0), OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then  ' the file could not be opened
   Debug.Print "Unable to open the file -- it probably does not exist."
   End  ' abort the program
End If

' Read a Long-type number from the file
retval = ReadFile(hFile, longbuffer, Len(longbuffer), numread, ByVal CLng(0))
If numread < Len(longbuffer) Then  ' EOF reached
   Debug.Print "End of file encountered -- could not read the data."
Else
   Debug.Print "Number read from file:"; longbuffer
End If

' Read a 10-character string from the file
stringbuffer = Space(10)  ' make room in the buffer
retval = ReadFile(hFile, ByVal stringbuffer, 10, numread, ByVal CLng(0))
If numread = 0 Then  ' EOF reached
Debug.Print "End of file encountered -- could not read any data."
ElseIf numread < 10 Then  ' read between 0 and 10 bytes
    Debug.Print "Incomplete string read: "; Left(stringbuffer, numread)
Else
    Debug.Print "String read from file: "; stringbuffer
End If

' Close the file.
retval = CloseHandle(hFile)

See Also

SetFilePointer, WriteFile

Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 13, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/readfile.html
Rectangle Function

Declare Function Rectangle Lib "gdi32.dll" (ByVal hdc As Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

Rectangle draws a rectangular-shaped box on a graphics-capable device. The rectangle is drawn in the device's current drawing color and is filled using its current filling color and brush, if any. The function returns 0 if an error occurred, or 1 if successful.

hdc
   The device context of the object to draw on.
X1
   The x coordinate of the rectangle's upper-left corner.
Y1
   The y coordinate of the rectangle's upper-left corner.
X2
   The x coordinate of the rectangle's lower-right corner.
Y2
   The y coordinate of the rectangle's lower-right corner.

Example:

' Draw a green rectangle on window Form1 with an upper-left
corner of (25,30) and a lower-right corner of (100,50)
Dim retval As Long ' return value
Form1.ForeColor = RGB(0, 255, 0) ' set the foreground-drawing color to green
retval = Rectangle (Form1.hdc, 25, 30, 100, 50) ' draw the rectangle from (25,30) -
(100,50)

See Also: RoundRect
Category: Filled Shapes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.

E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/rectangle.html
RectInRegion Function

Declare Function RectInRegion Lib "gdi32.dll" (ByVal hRgn As Long, lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

RectInRegion determines if a rectangle lies within a given region. The rectangle is considered to be inside the region if any portion of it -- not necessarily all of it -- lies within the region. The function returns 0 if the rectangle is completely outside the region, or a non-zero value if the rectangle is at least partially within the region.

**hRgn**
A handle to the region to determine if the rectangle lies within.

**lpRect**
The rectangle to determine if it lies at least partially within the region.

**Example:**

' Consider a line connecting the upper-right and lower-left corners of the ' screen, and consider the region made of the upper-left side of this line.
Determine ' if window Form1 at least partially lies within the region..
Dim swidth As Long, sheight As Long  ' width and height of the screen
Dim hRgn As Long  ' handle to the triangular region explained above
Dim winrect As RECT  ' receives Form1's rectangle
Dim vertices(0 To 2) As POINT_TYPE  ' vertices of region to create
Dim isinside As Long  ' receives 0 if not inside, non-zero otherwise
Dim retval As Long  ' generic return value

' Get the screen's width and height. Use this information to create the region.
swidth = GetSystemMetrics(SM_CXSCREEN)  ' screen width
sheight = GetSystemMetrics(SM_CYSCREEN)  ' screen height
' Load region's vertices into the array and create it.
vertices(0).x = 0: vertices(0).y = 0  ' vertex #1: upper-left corner of screen
vertices(1).x = swidth: vertices(1).y = 0  ' vertex #2: upper-right corner of screen
vertices(2).x = 0: vertices(2).y = sheight  ' vertex #3: lower-left corner of screen
hRgn = CreatePolygonRgn(vertices(0), 3, ALTERNATE)  ' create the region

' Get the rectangle of window Form1, identifying the corners of the window.
retval = GetWindowRect(Form1.hWnd, winrect)
' Determine if the rectangle lies within the region.
isinside = RectInRegion(hRgn, winrect)  ' is the rectangle in the region?
If isinside = 0 Then ' not inside
Debug.Print "Form1 is completely outside the region."
Else
    Debug.Print "Form1 lies at least partially inside the region."
End If

' Delete the region to free up resources.
retval = DeleteObject(hRgn)

See Also: PtInRegion
Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/rectinregion.html
recv Function

Declare Function recv Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

recv receives data from a connected socket. The socket must have already been connected to a network host via connect. If no response is heard from the network host, the function times out and reports an error.

Return Value

If successful, the function returns the number of bytes successfully read from the socket. If an error occurred (for example, if the connection timed out), the function returns SOCKET_ERROR (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

If specifying a string for buf, the ByVal keyword must be placed in front of it. See the example for a demonstration.

Parameters

s
A descriptor of the connected socket to receive data from.

buf
A buffer that receives the information obtained from the connection. This buffer must be large enough to receive the amount of data requested.

length
The size in bytes of the buffer passed as buf.

flags
A combination of the following flags specifying additional options:

- MSG_OOB
  Process out-of-band data only.
MSG_PEEK

Peek at the incoming data by copying data into the buffer without removing it from the input queue. If this flag is specified, the function returns the number of bytes of data currently pending to be received.

Constant Definitions

Const SOCKET_ERROR = -1
Const MSG_OOB = &H1
Const MSG_PEEK = &H2

Example

Download the main page of this web site (http://www.vbapi.com). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSADATA _
    As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
End Type
Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer
Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long
Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type
Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long
Declare Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, argp As Long) As Long
Public Const FIONBIO = &H8004667E
Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Const SOCKET_ERROR = -1
'
Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function
'
*** Place the following code inside the form window. ***
Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA  ' info about Winsock
    Dim sock As Long          ' the socket descriptor
    Dim pHostinfo As Long     ' pointer to info about the host computer
    Dim hostinfo As hostent   ' info about the host computer
    Dim pIPAddress As Long    ' pointer to host's IP address
    Dim ipAddress As Long     ' host's IP address
    Dim sockinfo As sockaddr  ' settings for the socket
    Dim buffer As String      ' buffer for sending and receiving data
    Dim reply As String       ' accumulates server's reply
    Dim retval As Long        ' generic return value

    ' Begin a Winsock session.

retval = **WSAStartup**(MAKEWORD(2, 2), wsockinfo)
If retval <> 0 Then
  Debug.Print "Unable to initialize Winsock! --"; retval
  Exit Sub
End If

' Get information about the server to connect to.
pHostinfo = **gethostbyname**("www.vbapi.com")
If pHostinfo = 0 Then
  Debug.Print "Unable to resolve host!"
  GoTo Cleanup
End If

' Copy information about the server into the structure.
**CopyMemory** hostinfo, ByVal pHostinfo, Len(hostinfo)
If hostinfo.h_addrtype <> AF_INET Then
  Debug.Print "Couldn't get IP address of www.vbapi.com!"
  GoTo Cleanup
End If

' Get the server's IP address out of the structure.
**CopyMemory** pIPAddress, ByVal hostinfo.h_addr_list, 4
**CopyMemory** ipAddress, ByVal pIPAddress, 4

' Create a socket.
sock = **socket**(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
  Debug.Print "Unable to create socket!"
  GoTo Cleanup
End If

' Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
  .sin_family = AF_INET
  ' Connect to port 80.
  .sin_port = **htons**(80)
  ' Connect to this IP address.
  .sin_addr = ipAddress
  ' Padding characters.
  .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = **connect**(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
  Debug.Print "Unable to connect!"
  GoTo Cleanup
End If

' Send an HTTP/GET request for the / document.
buffer = "GET / HTTP/1.1" & vbCrLf & _
  "Host: www.vbapi.com" & vbCrLf & _
  "User-Agent: HTTP-Test-Program" & vbCrLf & vbCrLf
retval = **send**(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request. Waiting for reply..."

' Make the socket non-blocking, so calls to recv don't halt the program waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)

' Read the response from the other system. A more sophisticated program would watch to see if the connection ever times out (i.e., if the connection is lost). For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If
    ' Process background events so the program doesn't appear to freeze.
    DoEvents
Loop Until retval = 0

' Print the response from the server.
Debug.Print "Document Retrieved:"
Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()
End Sub

See Also
recv

Category
Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/recv.html
RegCloseKey Function

Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegCloseKey closes a registry key that had previously been opened with RegCreateKeyEx or RegOpenKeyEx. Your program should use this function once it is finished using the registry key, in order to free resources. Obviously, you can no longer use the key after closing it.

Return Value

If successful, the function returns zero. If an error occurred, the function returns a non-zero error code.

Visual Basic-Specific Issues

None.

Parameters

hKey

A handle to the registry key to close.

Example

This example creates a new registry called "HKEY_CURRENT_USER\Software\MyCorp\MyProgram\Config". Under that key, it creates a "username" value and sets its value to the string "Rimmer". Place a command button named Command1 in a form window to run this example. The example executes when Command1 is clicked.

' This code is licensed according to the terms and conditions listed here.
Windows API Guide: RegCloseKey Function

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type SECURITY_ATTRIBUTES
  nLength As Long
  lpSecurityDescriptor As Long
  bInheritHandle As Long
End Type
Public Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA"
(ByVal _
  hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal lpClass
  As String, ByVal dwOptions As Long, ByVal samDesired As Long,
  lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As Long, lpdwDisposition As Long) As Long
Public Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA"
(ByVal _
  hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal dwType
  As Long, lpData As Any, ByVal cbData As Long) As Long
Public Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long
Public Const HKEY_CURRENT_USER = &H80000001
Public Const KEY_WRITE = &H20006
Public Const REG_SZ = 1
' *** Place the following code inside the form. ***
Private Sub Command1_Click()
  Dim hKey As Long ' receives handle to the registry key
  Dim secattr As SECURITY_ATTRIBUTES ' security settings for the key
  Dim subkey As String ' name of the subkey to create or open
  Dim neworused As Long ' receives flag for if the key was created or opened
  Dim stringbuffer As String ' the string to put into the registry
  Dim retval As Long ' return value

  ' Set the name of the new key and the default security settings
  subkey = "Software\MyCorp\MyProgram\Config"
  secattr.nLength = Len(secattr)
  secattr.lpSecurityDescriptor = 0
  secattr.bInheritHandle = 1

  ' Create (or open) the registry key.
  retval = RegCreateKeyEx(HKEY_CURRENT_USER, subkey, 0, "$", 0, KEY_WRITE, _
    secattr, hKey, neworused)
  If retval <> 0 Then
    Debug.Print "Error opening or creating registry key -- aborting."
    Exit Sub
  End If

  ' Write the string to the registry. Note the use of ByVal in the second-to-
  last ' parameter because we are passing a string.
stringbuffer = "Rimmer" & vbNullChar ' the terminating null is necessary
retval = RegSetValueEx(hKey, "username", 0, REG_SZ, ByVal stringbuffer, _
        Len(stringbuffer))

' Close the registry key.
retval = RegCloseKey(hKey)

End Sub

See Also

RegCreateKeyEx, RegOpenKeyEx

Category

Registry

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/regclosekey.html
RegCreateKeyEx Function

Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal lpClass As String, ByVal dwOptions As Long, ByVal samDesired As Long, lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As Long, lpdwDisposition As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

RegCreateKeyEx creates a new registry key. If the key you want to create already exists, the existing key will be opened (as if RegOpenKeyEx had been used). A handle to the created/opened key is put into the variable passed as phkResult.

Return Value

If successful, the function returns zero. If an error occurred, the function returns a non-zero error code.

Visual Basic-Specific Information

None.

Parameters

- **hKey**
  Specifies an existing registry under which the new registry key will be created. This is either a handle to an open registry key or exactly one of the following flags that identify a registry base key. (The base key flags are named identically to the base keys they specify.)
  - HKEY_CLASSES_ROOT
  - HKEY_CURRENT_CONFIG
  - HKEY_CURRENT_USER
  - HKEY_DYN_DATA (Windows 95, 98 only)
  - HKEY_LOCAL_MACHINE
  - HKEY_PERFORMANCE_DATA (Windows NT, 2000 only)
HKEY_USERS
lpSubkey
The name of the new key to create.
Reserved
Reserved -- set to 0.
lpClass
The name of the class or object type of the key. You can specify an empty string.
dwOptions
If this is 0, the key will be saved to the registry. If this is 1, the key will not be permanently saved to the registry, and will disappear when Windows shuts down.
samDesired
One or more of the following flags specifying the desired read/write access:
KEY_ALL_ACCESS
  Permission for all types of access.
KEY_CREATE_LINK
  Permission to create symbolic links.
KEY_CREATE_SUB_KEY
  Permission to create subkeys.
KEY_ENUMERATE_SUB_KEYS
  Permission to enumerate subkeys.
KEY_EXECUTE
  Same as KEY_READ.
KEY_NOTIFY
  Permission to give change notification.
KEY_QUERY_VALUE
  Permission to query subkey data.
KEY_READ
  Permission for general read access.
KEY_SET_VALUE
  Permission to set subkey data.
KEY_WRITE
  Permission for general write access.
lpSecurityAttributes
  Windows NT, 2000: Specifies the security attributes to assign to the newly created key. Windows 95, 98: Ignored.
phkResult
  Receives a handle to the newly created or opened registry key.
lpdwDisposition
  Receives 1 if the registry key was newly created, or 2 if an existing key was simply opened.

Constant Definitions

Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_CONFIG = &H80000005
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_DYN_DATA = &H80000006
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_PERFORMANCE_DATA = &H80000004
Const HKEY_USERS = &H80000003
Const KEY_ALL_ACCESS = &HF003F
Const KEY_CREATE_LINK = &H20
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_EXECUTE = &H20019
Const KEY_NOTIFY = &H10
Const KEY_QUERY_VALUE = &H1
Const KEY_READ = &H20019
Const KEY_SET_VALUE = &H2
Const KEY_WRITE = &H20006

Example

This example creates a new registry called "HKEY_CURRENT_USER\Software\MyCorp\MyProgram\Config". Under that key, it creates a "username" value and sets its value to the string "Rimmer". Place a command button named Command1 in a form window to run this example. The example executes when Command1 is clicked.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SECURITY_ATTRIBUTES
  nLength As Long
  lpSecurityDescriptor As Long
  bInheritHandle As Long
End Type
Public Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA"
  (ByVal _
   hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal lpClass
   As String, ByVal dwOptions As Long, ByVal samDesired As Long,
   lpSecurityAttributes _
   As SECURITY_ATTRIBUTES, phkResult As Long, lpdwDisposition As Long) As Long
Public Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA"
  (ByVal _
   hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal
   dwType As Long,lpData As Any, ByVal cbData As Long) As Long
Public Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long
Public Const HKEY_CURRENT_USER = &H80000001
Public Const KEY_WRITE = &H20006
Public Const REG_SZ = 1

' *** Place the following code inside the form. ***

Private Sub Command1_Click()
  Dim hKey As Long  ' receives handle to the registry key
  Dim secattr As SECURITY_ATTRIBUTES  ' security settings for the key
  Dim subkey As String  ' name of the subkey to create or open
  Dim neworused As Long  ' receives flag for if the key was created or opened
  Dim stringbuffer As String  ' the string to put into the registry
  Dim retval As Long  ' return value

  ' *** Place the following code inside the form. ***

' Set the name of the new key and the default security settings
subkey = "Software\MyCorp\MyProgram\Config"
secattr.nLength = Len(secattr)
secattr.lpSecurityDescriptor = 0
secattr.bInheritHandle = 1

' Create (or open) the registry key.
retval = RegCreateKeyEx(HKEY_CURRENT_USER, subkey, 0, "", 0, KEY_WRITE, _
                        secattr, hKey, neworused)
If retval <> 0 Then
    Debug.Print "Error opening or creating registry key -- aborting."
    Exit Sub
End If

' Write the string to the registry. Note the use of ByVal in the second-to-
last
' parameter because we are passing a string.
stringbuffer = "Rimmer" & vbNullChar ' the terminating null is necessary
retval = RegSetValueEx(hKey, "username", 0, REG_SZ, ByVal stringbuffer, _
                        Len(stringbuffer))

' Close the registry key.
retval = RegCloseKey(hKey)

End Sub

See Also
RegCloseKey, RegDeleteKey, RegOpenKeyEx

Category
Registry

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/regcreatekeyex.html
RegDeleteKey Function

Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" (ByVal hKey As Long, ByVal lpSubKey As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegDeleteKey deletes a key from the registry, along with any values it may contain. Windows NT, 2000: The function will fail if there are any subkeys under the key to delete. Windows 95, 98: The function will delete any subkeys under the key to be deleted, instead of failing.

Return Value

If successful, the function returns zero. If an error occurred, the function returns a nonzero error code.

Visual Basic-Specific Issues

None.

Parameters

hKey

Specifies an existing registry key that the key to delete is under. This is either a handle to an open registry key or one of the following flags that identify a registry base key. (The base key flags are named identically to the base keys they specify.)

- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
- HKEY_DYN_DATA (Windows 95, 98 only)
- HKEY_LOCAL_MACHINE
- HKEY_PERFORMANCE_DATA (Windows NT, 2000 only)
- HKEY_USERS
The name of the subkey under \textit{hKey} to delete.

**Constant Definitions**

`Const` \texttt{HKEY_CLASSES_ROOT} = \&H80000000

`Const` \texttt{HKEY_CURRENT_CONFIG} = \&H80000005

`Const` \texttt{HKEY_CURRENT_USER} = \&H80000001

`Const` \texttt{HKEY_DYN_DATA} = \&H80000006

`Const` \texttt{HKEY_LOCAL_MACHINE} = \&H80000002

`Const` \texttt{HKEY_PERFORMANCE_DATA} = \&H80000004

`Const` \texttt{HKEY_USERS} = \&H80000003

**Example:**

Attempt to delete the registry key "HKEY_LOCAL_MACHINE\Software\MyProgram\Config" when the user clicks button Command1. (Of course, typically you would never give the user an option like this, but this is just a simple example.) Depending on your Windows platform, this function may fail if there are any subkeys under this key. To run this example, you first need to place a command button named Command1 on a form window.

* This code is licensed according to the terms and conditions listed [here](#).

* Declarations and such needed for the example:
  * (Copy them to the (declarations) section of a module.)

```vba
Public Declare Function RegDeleteKey Lib "advapi32.dll" Alias "RegDeleteKeyA" (ByVal hKey As Long, ByVal lpSubKey As String) As Long

Public Const HKEY_LOCAL_MACHINE = &H80000002

Private Sub Command1_Click()
    Dim retval As Long  ' return value

    ' Attempt to delete the desired registry key.
    retval = RegDeleteKey(HKEY_LOCAL_MACHINE, "Software\MyProgram\Config")
    If retval <> 0 Then
        Debug.Print "Delete operation failed. The key may have subkeys and/or values under it."
    Else
        Debug.Print "Of course, the key might just not exist at all."
    End If
End Sub
```

**See Also**

RegCreateKeyEx, RegDeleteValue

**Category**
RegDeleteValue Function

Declare Function RegDeleteValue Lib "advapi32.dll" Alias "RegDeleteValueA" (ByVal hKey As Long, ByVal lpValueName As String) As Long

Platforms: Win 95/98, Win NT

RegDeleteValue deletes a value stored under a specified key in the registry. This function only works with values stored; it cannot delete subkeys. The value can of course be of any registry data type. The function returns 0 if successful, or a non-zero error code if an error occured.

**hKey**
A handle to the open registry key which contains the value to delete.

**lpValueName**
The name of the value to delete.

**Example:**

' Delete the value "SplashScreen" under the hypothetical registry key
' "HKEY_LOCAL_MACHINE\Software\MyProgram\Config". Note how error conditions are checked.
Dim hkey As Long  ' handle to the open registry key
Dim retval As Long  ' return value

' First, open up the registry key which holds the value to delete.
retval = RegOpenKeyEx(HKEY_LOCAL_MACHINE, "Software\MyProgram\Config", 0, KEY_ALL_ACCESS, hkey)
If retval = 0 Then  ' successfully opened registry key
  ' Now delete the desired value from the key.
  retval = RegDeleteValue(hkey, "SplashScreen")  ' if it existed, it is now deleted
  ' Note that we only have to close the registry key if it was successfully opened.
  retval = RegCloseKey
End If

See Also: RegDeleteKey, RegQueryValueEx, RegSetValueEx

Category: Registry

Go back to the alphabetical Function listing.
Go back to the Reference section index.
RegEnumKeyEx Function

Declare Function RegEnumKeyEx Lib "advapi32.dll" Alias "RegEnumKeyExA" (ByVal hKey As Long, ByVal dwIndex As Long, ByVal lpName As String, lpcbName As Long, lpReserved As Long, ByVal lpClass As String, lpcbClass As Long, lpftLastWriteTime As FILETIME) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegEnumKeyEx enumerates all of the subkeys under a given key. The function retrieves the subkey name, class name, and last write time of each subkey. The key under which the subkeys are enumerated must have been opened with subkey-enumeration access (see the example). The program must use the function in a loop, incrementing the index value (which determines which subkey is identified) until the list has been exhausted. The subkeys are not retrieved in any clear order.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

hKey
A handle to the open registry key containing the subkeys to be enumerated (having been opened with subkey-enumeration access). This could also be one of the following flags identifying a predefined base keys:

- HKEY_CLASSES_ROOT
  The HKEY_CLASSES_ROOT base key.
- HKEY_CURRENT_CONFIG
  The HKEY_CURRENT_CONFIG base key.
- HKEY_CURRENT_USER
  The HKEY_CURRENT_USER base key.
The HKEY_CURRENT_USER base key.

HKEY_DYN_DATA
Windows 95, 98: The HKEY_DYN_DATA base key.

HKEY_LOCAL_MACHINE
The HKEY_LOCAL_MACHINE base key.

HKEY_PERFORMANCE_DATA
Windows NT, 2000: The HKEY_PERFORMANCE_DATA base key.

HKEY_USERS
The HKEY_USERS base key.

dwIndex
The index of the particular subkey to retrieve information about. Valid indices begin with 0 and go up to one less than the number of subkeys.

lpName
String which receives the name of the subkey whose information is being retrieved. This must be initialized to a sufficient size to receive the string.

lpbName
The size of the string passed as lpName. This also receives the length of the string the function places in lpName.

lpReserved
Reserved -- set to 0. Visual Basic users must use the ByVal keyword immediately before the 0.

lpClass
String which receives the name of the subkey's class. This must be initialized to a sufficient size to receive the string.

lpbClass
The size of the string passed as lpClass. This also receives the length of the string the function places in lpClass.

lpftLastWriteTime
Receives the time and date on which the subkey was last written to.

Constant Definitions

Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_CONFIG = &H80000005
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_DYN_DATA = &H80000006
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_PERFORMANCE_DATA = &H80000004
Const HKEY_USERS = &H80000003

Example

' This code is licensed according to the terms and conditions listed here.
'
' Enumerate the subkeys under HKEY_LOCAL_MACHINE\Software.  The name
' and class of each subkey is displayed for the user.  Note the use of the loop which
' starts at 0 and keeps incrementing the index until no more subkeys exist.
Dim keyname As String  ' receives name of each subkey
Dim keylen As Long  ' length of keyname
Dim classname As String  ' receives class of each subkey
Dim classlen As Long  ' length of classname
Dim lastwrite As FILETIME  ' receives last-write-to time, but we ignore it here
Dim hkey As Long  ' handle to the HKEY_LOCAL_MACHINE\Software key
Dim index As Long  ' counter variable for index
Dim retval As Long  ' function's return value

' Open the desired registry key. Note the access level requested.
retval = RegOpenKeyEx(HKEY_LOCAL_MACHINE, "Software", 0, KEY_ENUMERATE_SUB_KEYS, hkey)

' Test to make sure the key was opened successfully.
If retval <> 0 Then
    Debug.Print "Registry key could not be opened -- aborting."
    End ' terminate the program
End If

' List through each possible subkey. Note how the strings receiving the information 
' must be reinitialized each loop iteration.
index = 0  ' initial index value
While retval = 0  ' while we keep having success (retval equals 0 from the above API call)
    keyname = Space(255): classname = Space(255)  ' make room in string buffers
    keylen = 255: classlen = 255  ' identify the allocated space
    ' Get information about the next subkey, if one exists.
    retval = RegEnumKeyEx(hkey, index, keyname, keylen, ByVal 0, classname, classlen, lastwrite)
    If retval = 0  ' only display info if another subkey was found
        ' Extract the useful information from the string buffers.
        keyname = Left(keyname, keylen)  ' trim off the excess space
        classname = Left(classname, classlen)
        ' Display the returned information.
        Debug.Print "HKEY_LOCAL_MACHINE\Software\"; keyname  ' display full subkey name
        Debug.Print "  (class: "; classname  ' display subkey's class
    End If
    index = index + 1  ' increment the index counter
Wend  ' end the loop

' Close the registry key after enumeration is complete.
retval = RegCloseKey(hkey)

See Also
RegEnumValue

Category
Registry

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: March 19, 2000
RegEnumValue Function

Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" (ByVal hKey As Long, ByVal dwIndex As Long, ByVal lpValueName As String, lpcbValueName As Long, ByVal lpReserved As Long, lpType As Long, lpData As Byte, lpcbData As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegEnumValue enumerates all of the values under a given registry key. The function must be used in a loop because each call only retrieves information about the value identified by the given index. Besides merely listing the names of each value, the function can also retrieve the data associated with each value. This data, which could be in any valid registry data format, is placed in an array of bytes passed as lpData. It is the program's responsibility to correctly interpret the information. The key whose values are enumerated must have been opened with query-level access. The values which are enumerated are not retrieved in any obvious order.

Return Value

If successful, the function returns 0. If an error occurred, the function returns a non-zero error code.

Visual Basic-Specific Issues

When passing 0 for lpData or lpcbData, the ByVal keyword must be placed in front if the 0 (i.e., use ByVal 0 instead of just 0).

Parameters

hKey

A handle to the opened registry key containing the values to enumerate (having been opened with query-level access). This could also be exactly one of the following base keys (identified by identically named flags):
- HKEY_CLASSES_ROOT
- HKEY_CURRENT_CONFIG
- HKEY_CURRENT_USER
HKEY_DYN_DATA (Windows 95, 98 only)
HKEY_LOCAL_MACHINE
HKEY_PERFORMANCE_DATA (Windows NT, 2000 only)
HKEY_USERS

dwIndex
The index of the particular value to retrieve information about. Valid indices range from 0 to one less than the total number of values.

lpValueName
String which receives the name of the value whose information is being retrieved. This must be initialized to a sufficient size to receive the string.

lpcbValueName
The size of the string passed as lpValueName. This also receives the size of the string the function places inside of it.

lpReserved
Reserved -- set to 0.

lpType
Receives exactly one of the following flags identifying the data type of the value:
REG_BINARY
A non-text sequence of bytes.
REG_DWORD
Same as REG_DWORD_LITTLE_ENDIAN.
REG_DWORD_BIG_ENDIAN
A 32-bit integer stored in big-endian format. This is the opposite of the way Intel-based computers normally store numbers -- the byte order is reversed, putting the most significant byte first in memory.
REG_DWORD_LITTLE_ENDIAN
A 32-bit integer stored in little-endian format. This is the way Intel-based computers store numbers, putting the least significant byte first in memory.
REG_EXPAND_SZ
A null-terminated string which contains unexpanded environment variables.
REG_LINK
A Unicode symbolic link.
REG_MULTI_SZ
A series of strings, each separated by a null character and the entire set terminated by a two null characters.
REG_NONE
No data type.
REG_RESOURCE_LIST
A list of resources in the resource map.
REG_SZ
A string terminated by a null character.

lpData
A byte array receiving the data stored in the value. If you do not want to retrieve the data, set this parameter to 0.

lpcbData
The size in bytes of the array passed as lpData. This also receives the size in bytes of the data placed into the array. If you do not want to retrieve any data, set this parameter to 0.

Constant Definitions

Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_CONFIG = &H80000005
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_DYN_DATA = &H80000006
Const HKEY_LOCAL_MACHINE = &H80000002
Windows API Guide: RegEnumValue Function

Const HKEY_PERFORMANCE_DATA = &H80000004
Const HKEY_USERS = &H80000003
Const REG_BINARY = 3
Const REG_DWORD = 4
Const REG_DWORD_BIG_ENDIAN = 5
Const REG_DWORD_LITTLE_ENDIAN = 4
Const REG_EXPAND_SZ = 2
Const REG_LINK = 6
Const REG_MULTI_SZ = 7
Const REG_NONE = 0
Const REG_RESOURCE_LIST = 8
Const REG_SZ = 1

Example

List the values under the registry key HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion. Display the names of all values under the key. Also display the values' data if they are either null-terminated strings or binary data. (This example could be extended to include the rest of the registry data types, but it doesn't to save space). Note how the byte array is used to buffer the data before it might be copied to the string.

The example runs when the user clicks button Command1. To use this example, then, you must first place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function RegEnumValue Lib "advapi32.dll" Alias "RegEnumValueA" (ByVal hKey As Long, ByVal dwIndex As Long, ByVal lpValueName As String, lpcbValueName As Long, ByVal lpReserved As Long, lpType As Long, lpData As Byte, lpcbData As Long) As Long
Public Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, phkResult As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Const HKEY_LOCAL_MACHINE = &H80000002
Public Const KEY_QUERY_VALUE = &H1
Public Const REG_BINARY = 3

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim valuename As String     ' name of the value being retrieved

    ' Example code starts here...

    ' Example code ends here...

    ' *** Place this code inside a form window. ***

    ' Example code ends here...

Dim valuelen As Long        ' length of valuename
Dim datatype As Long        ' receives data type of value
Dim data(0 To 254) As Byte  ' 255-byte data buffer for read information
Dim datalen As Long         ' size of data buffer information
Dim datastring As String    ' will receive data converted to a string, if necessary
Dim hkey As Long            ' handle to the registry key to enumerate the values of
Dim index As Long           ' counter for the index of the value to enumerate
Dim c As Long               ' counter variable
Dim retval As Long          ' functions' return value

' Open the registry key to enumerate the values of.
retval = RegOpenKeyEx(HKEY_LOCAL_MACHINE,
"Software\Microsoft\Windows\CurrentVersion", _
0, KEY_QUERY_VALUE, hkey)
' Check to see if an error occurred.
If retval <> 0 Then  
    Debug.Print "Registry key could not be opened -- aborting."
    End  ' abort the program
End If

' Begin enumerating the values. Get each one, displaying its name. If it's a null-
' terminated string or binary data, display it. If not, say so.
index = 0  ' initialize the counter
While retval = 0  ' loop while successful
    ' Initialize the value name buffer.
    valuename = Space(255)  ' 255-space buffer
    valuelen = 255  ' length of the string
    datalen = 255  ' size of data buffer
    ' Get the next value to be enumerated
    retval = RegEnumValue(hkey, index, valuename, valuelen, 0, datatype, data(0), datalen)
    If retval = 0 Then  ' if successful, display information
        ' Extract the useful information from the value name buffer and display it.
        valuename = Left(valuename, valuelen)
        Debug.Print "Value Name: "; valuename
        ' Determine the data type of the value and display it.
        Select Case datatype
            Case REG_SZ  ' null-terminated string
                ' Copy the information from the byte array into the string.
                datastring = Space(datalen - 1)  ' make just enough room in the string
                CopyMemory ByVal datastring, data(0), datalen - 1  ' copy useful data
                Debug.Print "  Data (string): "; datastring
            Case REG_BINARY  ' binary data
...
' Display the hexadecimal values of each byte of data, separated by spaces. Use the datastring buffer to allow us to assure each byte is represented by a two-character string.

Debug.Print "  Data (binary):";
For c = 0 To datalen - 1 ' loop through returned information
datastring = Hex(data(c)) ' convert value into hex
' If needed, add leading zero(s).
If Len(datastring) < 2 Then datastring = _
          String(2 - Len(datastring), "0") &
datastring
        Debug.Print " "; datastring;
        Next c
        Debug.Print ' end the line
        Case Else ' a data type this example doesn't handle
        Debug.Print "This example doesn't know how to read that kind of data."
        End Select
        End If
        index = index + 1 ' increment the index counter
        Wend ' end the loop
'
' Close the registry key.
   retval = RegCloseKey(hkey)

End Sub

See Also

RegEnumKeyEx

Category

Registry

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/regenumvalue.html
RegisterClass Function

Declare Function RegisterClass Lib "user32.dll" Alias "RegisterClassA" (lpWndClass As WNDCLASS) As Long

Platforms

- **Windows 95**: Supported but Obsolete; use `RegisterClassEx` instead.
- **Windows 98**: Supported but Obsolete; use `RegisterClassEx` instead.
- **Windows NT**: Requires Windows NT 3.1 or later but Obsolete in Windows NT 4.0 or later; use `RegisterClassEx` instead.
- **Windows 2000**: Supported but Obsolete; use `RegisterClassEx` instead.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

`RegisterClass` registers a new window class for use. Only after registering the window class can it be used to create windows. This function is unable to set the small icon associated with a window class; to do that, use the more powerful `RegisterClassEx` function. After the window class is completely finished being used, use `UnregisterClass` to unregister it.

Return Value

If an error occurred, the function returns 0 (use `GetLastError` to get the error code). If successful, the function returns an atom identifying the class.

Visual Basic-Specific Issues

None.

Parameters

`lpWndClass`

Information about the window class being registered.

Example

' This code is licensed according to the terms and conditions listed here.
' Register a new window class. The various properties to
give it will be explained in the code.

' *** Place the following code in a module. ***
' Define the window procedure to use for the class. Here, we'll
' just make a wrapper for the default window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
' Return whatever the default window procedure returns.
WindowProc = DefWindowProc(hWnd, uMsg, wParam, lParam)
End Function

' *** Place the following code where you want to register the class. ***
Dim classinfo As WNDCLASS ' holds info about the class
Dim classatom As Long ' receives an atom to the class just registered

' Set the various data members of the structure.
' Class style: give each window its own DC; redraw when resized.
classinfo.style = CS_OWNDC Or CS_HREDRAW Or CS_VREDRAW
' Use the window procedure above to process messages.
classinfo.lpfnWndProc = AddressOf WindowProc
' We aren't using any extra information.
classinfo.cbClsExtra = 0
classinfo.cbWndExtra = 0
' Handle to the instance of this application.
classinfo.hInstance = App.hInstance
' Use the icon stored in C:\MyApp\deficon.ico.
classinfo.hIcon = ExtractIcon(App.hInstance, "C:\MyApp\deficon.ico", 0)
' Use the cursor stored in C:\MyApp\mouse.cur.
classinfo.hCursor = LoadCursorFromFile("C:\MyApp\mouse.cur")
' Fill the background with the system color for an application's workspace.
classinfo.hbrBackground = COLOR_APPWORKSPACE
' No menu resource to use.
classinfo.lpszMenuName = ""
' Give the class a name.
classinfo.lpszClassName = "CustomClass"

' Finally, register the class.
classatom = RegisterClass(classinfo)
' Now the class CustomClass can be used to create windows.

' *** Place the following code where you wish to unregister the window class. ***
Dim retval As Long

' Unregister the window class.
retval = UnregisterClass("CustomClass", App.hInstance)

See Also
RegisterClassEx Function

Declare Function RegisterClassEx Lib "user32.dll" Alias "RegisterClassExA" (lpwcx As WNDCLASSEX) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

RegisterClassEx registers a new window class for use. Only after registering the window class can it be used to create windows. After the window class is completely finished being used, use UnregisterClass to unregister it.

Return Value

If an error occurred, the function returns 0. If successful, the function returns an atom identifying the class.

Visual Basic-Specific Issues

None.

Parameters

lpwcx

Information about the window class being registered.

Example

Register a new window class for later use. Although this example doesn't actually use the newly created class, it shows how this class would be made and, later, destroyed. The code executes in the Load and Unload logic of a form window Form1.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WNDCLASSEX
    cbSize As Long
    style As Long
    lpfnWndProc As Long
    cbClsExtra As Long
    cbWndExtra As Long
    hInstance As Long
    hIcon As Long
    hCursor As Long
    hbrBackground As Long
    lpszMenuName As String
    lpszClassName As String
    hIconSm As Long
End Type
Public Const CS_OWNDC = &H20
Public Const CS_HREDRAW = &H2
Public Const CS_VREDRAW = &H1
Public Const COLOR_APPWORKSPACE = 12
Public Declare Function RegisterClassEx Lib "user32.dll" Alias "RegisterClassExA" _
            (lpwcx As WNDCLASSEX) As Long
Public Declare Function UnregisterClass Lib "user32.dll" Alias "UnregisterClassA" _
            (ByVal lpClassName As Any, ByVal hInstance As Long) As Long
Public Declare Function LoadCursorFromFile Lib "user32.dll" Alias "LoadCursorFromFileA" _
            (ByVal lpFileName As String) As Long
Public Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA" (ByVal hInst _
            As Long, ByVal lpzExeFileName As String, ByVal nIconIndex As Long) As Long

' *** Place the following code in a module. ***

' Define the window procedure to use for the class. Here, we'll
' just make a wrapper for the default window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, _
             ByVal lParam As Long) As Long
    ' Return whatever the default window procedure returns.
    WindowProc = DefWindowProc(hWnd, uMsg, wParam, lParam)
End Function

' The following function returns the value passed to it. This allows the
' example to use the AddressOf operator to set a value in the structure.
Public Function DummyFunc (ByVal dwValue As Long) As Long
    DummyFunc = dwValue
End Function

' *** Place the following code inside window Form1. ***
Private Sub Form_Load()
Dim classinfo As WNDCLASSEX  ' holds info about the class
Dim classatom As Long        ' receives an atom to the class just registered

' Load a description of the new class into the structure.
With classinfo
  ' Of course, set the size of the structure.
  .cbSize = Len(classinfo)
  ' Class style: give each window its own DC; redraw when resized.
  .style = CS_OWNDC Or CS_HREDRAW Or CS_VREDRAW
  ' Use the window procedure above to process messages.
  .lpfnWndProc = DummyFunc(AddressOf WindowProc)
  ' We aren't using any extra information.
  .cbClsExtra = 0
  .cbWndExtra = 0
  ' Handle to the instance of this application.
  .hInstance = App.hInstance
  ' Use the icon stored in C:\MyApp\deficon.ico.
  .hIcon = ExtractIcon(App.hInstance, "C:\MyApp\deficon.ico", 0)
  ' Use the cursor stored in C:\MyApp\mouse.cur.
  .hCursor = LoadCursorFromFile("C:\MyApp\mouse.cur")
  ' Fill the background with the system color for an application's workspace.
  .hbrBackground = COLOR_APPWORKSPACE
  ' No menu resource to use.
  .lpszMenuName = ""
  ' Give the class a name.
  .lpszClassName = "CustomClass"
  ' Use the small icon stored in C:\MyApp\deficonsm.ico.
  .hIconSm = ExtractIcon(App.hInstance, "C:\MyApp\deficonsm.ico", 0)
End With

' Finally, register the class.
classatom = RegisterClassEx(classinfo)
' Now the class CustomClass can be used to create windows.

End Sub

Private Sub Form_Unload(Cancel As Integer)
Dim retval As Long  ' return value

' Unregister the window class when the form closes.
retval = UnregisterClass("CustomClass", App.hInstance)

End Sub

See Also

RegisterClass, UnregisterClass

Category
Windows API Guide: RegisterClassEx Function

Window Classes

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/registerclassex.html
RegOpenKeyEx Function

Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long, phkResult As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

RegOpenKeyEx opens a key in the Windows registry. The handle it gives must be used when you read to or write from any values under that key. Unlike RegCreateKeyEx, this function will not create the key if it does not exist. The function puts a handle to the opened key into the variable passed as phkResult. The function returns 0 if successful, or a non-zero value error code if an error occurred.

hKey
Either the handle to an open registry key or exactly one of the following flags that the desired key is under:
HKEY_CURRENT_USER = &H80000001
The HKEY_CURRENT_USER base key, which stores program information for the current user.
HKEY_LOCAL_MACHINE = &H80000002
The HKEY_LOCAL_MACHINE base key, which stores program information for all users.
HKEY_USERS = &H80000003
The HKEY_USERS base key, which has all the information for any user (not just the one provided by HKEY_CURRENT_USER).
HKEY_CURRENT_CONFIG = &H80000005
The HKEY_CURRENT_CONFIG base key, which stores computer configuration information.
HKEY_DYN_DATA = &H80000006
The HKEY_DYN_DATA base key, which stores dynamic data.

lpSubKey
The name of the key to open.

ulOptions
Reserved. Set to 0.

samDesired
One or more of the following flags specifying the desired read/write access:
KEY_ALL_ACCESS = &HF003F
Permission for all types of access.
KEY_CREATE_LINK = &H20
Permission to create symbolic links.
KEY_CREATE_SUB_KEY = &H4
Permission to create subkeys.
KEY_ENUMERATE_SUB_KEYS = &H8
Permission to enumerate subkeys.
KEY_EXECUTE = &H20019
    Same as KEY_READ.
KEY_NOTIFY = &H10
    Permission to give change notification.
KEY_QUERY_VALUE = &H1
    Permission to query subkey data.
KEY_READ = &H20019
    Permission for general read access.
KEY_SET_VALUE = &H2
    Permission to set subkey data.
KEY_WRITE = &H20006
    Permission for general write access.

phkResult
    Receives the handle to the registry key.

Example:

' Open a key called HKEY_CURRENT_USER\Software\MyCorp\MyProgram\Config.
' Then create a "username" value under that key and set its value to "Rimmer".
Dim hregkey As Long  ' receives handle to the opened registry key
Dim subkey As String  ' name of the subkey to create
Dim retval As Long  ' return value

' Set the name of the new key and the default security settings
subkey = "Software\MyCorp\MyProgram\Config"

' Open the registry key
retval = RegOpenKeyEx(HKEY_CURRENT_USER, subkey, 0, KEY_WRITE, hregkey)
If retval <> 0 Then  ' error during open
    Debug.Print "Error opening registry key -- aborting."
    End  ' terminate the program
End If

' Insert rest of code here.....

' Close the registry key
retval = RegCloseKey(hregkey)

See Also: RegCloseKey, RegCreateKeyEx
Category: Registry

Go back to the alphabetical Function listing.
Go back to the Reference section index.
RegQueryValueEx Function

Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, lpType As Long, lpData As Any, lpcbData As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegQueryValueEx reads a value from a registry key. It can read many different types of data, including integers, strings, and any other registry data types. When calling the function, the program does not have to know what the data type of the value being read is. Instead, the program receives information telling it what type of data was read.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

When putting the data read from the registry into a string, the lpData parameter must be prefixed by the ByVal keyword. The ByVal keyword is not necessary with any other data types passed as that parameter.

Parameters

hKey

A handle to the registry key to read the value from. This could also be one of the following flags identifying one of the predefined registry base keys. The flags have identical names to the registry base keys they specify.

HKEY_CLASSES_ROOT
HKEY_CURRENT_CONFIG
HKEY_CURRENT_USER
HKEY_DYN_DATA (Windows 95, 98 only)
HKEY_LOCAL_MACHINE
HKEY_PERFORMANCE_DATA (Windows NT, 2000 only)
HKEY_USERS

lpValueName
The name of the value to read.

Reserved
Reserved. Set to 0.

lpType
Receives one of the following flags identifying the data type of the data being read:

- REG_BINARY
  A non-text sequence of bytes.

- REG_DWORD
  Same as REG_DWORD_LITTLE_ENDIAN.

- REG_DWORD_BIG_ENDIAN
  A 32-bit integer stored in big-endian format. This is the opposite of the way Intel-based computers normally store numbers -- the byte order is reversed.

- REG_DWORD_LITTLE_ENDIAN
  A 32-bit integer stored in little-endian format. This is the way Intel-based computers store numbers.

- REG_EXPAND_SZ
  A null-terminated string which contains unexpanded environment variables.

- REG_LINK
  A Unicode symbolic link.

- REG_MULTI_SZ
  A series of strings, each separated by a null character and the entire set terminated by a two null characters.

- REG_NONE
  No data type.

- REG_RESOURCE_LIST
  A list of resources in the resource map.

- REG_SZ
  A string terminated by a null character.

lpData
Variable, array, or some other object that receives the information read from the registry.

lpcbData
Set this to the length in bytes of whatever was passed as lpData to receive the data read from the registry. This parameter also receives the length in bytes of the data actually read from the registry.

## Constant Definitions

```plaintext
Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_CONFIG = &H80000005
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_DYN_DATA = &H80000006
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_PERFORMANCE_DATA = &H80000004
Const HKEY_USERS = &H80000003
Const REG_BINARY = 3
Const REG_DWORD = 4
Const REG_DWORD_BIG_ENDIAN = 5
Const REG_DWORD_LITTLE_ENDIAN = 4
Const REG_EXPAND_SZ = 2
Const REG_LINK = 6
Const REG_MULTI_SZ = 7
Const REG_NONE = 0
```

Windows API Guide: RegQueryValueEx Function

Const REG_RESOURCE_LIST = 8
Const REG_SZ = 1

Example

Open the registry key HKEY_CURRENT_USER\Software\MyCorp\MyProgram\Config and read the value named "username" stored in it. The example given for the RegSetValueEx function creates this key and sets the value appropriately. The example runs when button Command1 is pressed. To run this example, you need to place a command button named Command1 inside a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function RegOpenKeyEx Lib "advapi32.dll" Alias "RegOpenKeyExA" (ByVal _
    hKey As Long, ByVal lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired _
    As Long, phkResult As Long) As Long
Public Declare Function RegQueryValueEx Lib "advapi32.dll" Alias "RegQueryValueExA" (ByVal _
    hKey As Long, ByVal lpValueName As String, ByVal lpReserved As Long, ByVal lpType As Long, _
    lpData As Any, lpcbData As Long) As Long
Public Declare Function RegCloseKey Lib "advapi32.dll" (ByVal hKey As Long) As Long
Public Const HKEY_CURRENT_USER = &H80000001
Public Const KEY_READ = &H20019
Public Const REG_SZ = 1

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim hKey As Long  ' receives a handle to the newly created or opened registry key
    Dim subkey As String  ' name of the subkey to open
    Dim stringbuffer As String  ' receives data read from the registry
    Dim datatype As Long  ' receives data type of read value
    Dim slength As Long  ' receives length of returned data
    Dim retval As Long  ' return value

    ' Set the name of the new key and the default security settings
    subkey = "Software\MyCorp\MyProgram\Config"

    ' Create or open the registry key
    retval = RegOpenKeyEx(HKEY_CURRENT_USER, subkey, 0, KEY_READ, hKey)
    If retval <> 0 Then
        Debug.Print "ERROR: Unable to open registry key!"
        Exit Sub
    End If

    ' Make room in the buffer to receive the incoming data.
    stringbuffer = Space(255)
    slength = 255
' Read the "username" value from the registry key.
retval = RegQueryValueEx(hKey, "username", 0, datatype, ByVal stringbuffer, slength)

' Only attempt to display the data if it is in fact a string.
If datatype = REG_SZ Then
    ' Remove empty space from the buffer and display the result.
    stringbuffer = Left(stringbuffer, slength - 1)
    Debug.Print "Username: "; stringbuffer
Else
    ' Don't bother trying to read any other data types.
    Debug.Print "Data not in string format. Unable to interpret data."
End If

' Close the registry key.
retval = RegCloseKey(hKey)

End Sub

See Also

RegDeleteValue, RegSetValueEx

Category

Registry

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/regqueryvalueex.html
RegSetValueEx Function

Declare Function RegSetValueEx Lib "advapi32.dll" Alias "RegSetValueExA" (ByVal hKey As Long, ByVal lpValueName As String, ByVal Reserved As Long, ByVal dwType As Long, lpData As Any, ByVal cbData As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RegSetValueEx writes a value to a registry key. If the value does not already exist, it will be created. The value can be of any of the registry data types.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

When writing a string or a single numeric value, the lpData parameter must be prefixed by the ByVal keyword. Any other values (such as byte arrays) do not need the ByVal keyword.

Parameters

hKey

A handle to the registry key to write the value under. This could also be one of the following flags identifying one of the predefined registry base keys:

- HKEY_CLASSES_ROOT
  The HKEY_CLASSES_ROOT base key.
- HKEY_CURRENT_CONFIG
  The HKEY_CURRENT_CONFIG base key.
- HKEY_CURRENT_USER
  The HKEY_CURRENT_USER base key.
- HKEY_DYN_DATA
  The HKEY_DYN_DATA base key.
Windows 95, 98: The HKEY_DYN_DATA base key.
HKEY_LOCAL_MACHINE
  The HKEY_LOCAL_MACHINE base key.
HKEY_PERFORMANCE_DATA
Windows NT, 2000: The HKEY_PERFORMANCE_DATA base key.
HKEY_USERS
  The HKEY_USERS base key.

*lpValueName*

The name of the value to set. If this is an empty string, the function writes to the unnamed default value (*Windows 95:*
the default value always has the REG_SZ data type).

*Reserved*

Reserved. Set to 0.

*dwType*

One of the following flags identifying the data type of the data to write to the registry:
- **REG_BINARY**
  A non-text sequence of bytes.
- **REG_DWORD**
  Same as REG_DWORD_LITTLE_ENDIAN.
- **REG_DWORD_BIG_ENDIAN**
  A 32-bit integer stored in big-endian format. This is the opposite of the way Intel-based computers normally store numbers -- the word order is reversed.
- **REG_DWORD_LITTLE_ENDIAN**
  A 32-bit integer stored in little-endian format. This is the way Intel-based computers normally store numbers.
- **REG_EXPAND_SZ**
  A null-terminated string which contains unexpanded environment variables.
- **REG_LINK**
  A Unicode symbolic link.
- **REG_MULTI_SZ**
  A series of strings, each separated by a null character and the entire set terminated by a two null characters.
- **REG_NONE**
  No data type.
- **REG_RESOURCE_LIST**
  A list of resources in the resource map.
- **REG_SZ**
  A string terminated by a null character.

*lpData*

The number, string, or other data to write to the registry.

*cbData*

The size in bytes of the data being written to the registry.

**Constant Definitions**

```
Const HKEY_CLASSES_ROOT = &H80000000
Const HKEY_CURRENT_CONFIG = &H80000005
Const HKEY_CURRENT_USER = &H80000001
Const HKEY_DYN_DATA = &H80000006
Const HKEY_LOCAL_MACHINE = &H80000002
Const HKEY_PERFORMANCE_DATA = &H80000004
Const HKEY_USERS = &H80000003
Const REG_BINARY = 3
Const REG_DWORD = 4
```
Const REG_DWORD_BIG_ENDIAN = 5
Const REG_DWORD_LITTLE_ENDIAN = 4
Const REG_EXPAND_SZ = 2
Const REG_LINK = 6
Const REG_MULTI_SZ = 7
Const REG_NONE = 0
Const REG_RESOURCE_LIST = 8
Const REG_SZ = 1

Example

' This code is licensed according to the terms and conditions listed here.

' Create a key called HKEY_CURRENT_USER\Software\MyCorp\MyProgram\Config.
' Then create a "username" value under that key and set its value to "Rimmer".
Dim hregkey As Long  ' receives handle to the newly created or opened registry key
Dim secattr As SECURITY_ATTRIBUTES  ' security settings of the key
Dim subkey As String  ' name of the subkey to create
Dim neworused As Long  ' receives 1 if new key was created or 2 if an existing key was opened
Dim stringbuffer As String  ' the string to put into the registry
Dim retval As Long  ' return value

' Set the name of the new key and the default security settings
subkey = "Software\MyCorp\MyProgram\Config"
secattr.nLength = Len(secattr)  ' size of the structure
secattr.lpSecurityDescriptor = 0  ' default security level
secattr.bInheritHandle = True  ' the default value for this setting

' Create or open the registry key
retval = RegCreateKeyEx(HKEY_CURRENT_USER, subkey, 0, "", 0, KEY_WRITE, secattr, hregkey, neworused)
If retval <> 0 Then  ' error during open
    Debug.Print "Error opening or creating registry key -- aborting."
    End  ' terminate the program
End If

' Write the string to the registry.  Note that because Visual Basic is being used, the string
' passed to the function must explicitly be passed ByVal.
stringbuffer = "Rimmer" & vbNullChar  ' note how a null character must be appended to the string
retval = RegSetValueEx(hregkey, "username", 0, REG_SZ, ByVal stringbuffer, Len(stringbuffer))  ' write the string

' Close the registry key
retval = RegCloseKey(hregkey)

See Also

RegDeleteValue, RegQueryValueEx

Category

Registry

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 11, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/regsetvalueex.html
ReleaseCapture Function

Declare Function ReleaseCapture Lib "user32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

ReleaseCapture ends whatever mouse capture may be in effect, regardless of which window actually captured the mouse. Once ReleaseCapture is called, mouse messages are immediately routed back to the windows that would normally receive them. This function should be used as soon as a window no longer needs to capture the mouse after it had called SetCapture.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

None.

Example

The following example assumes that there is a picture box control, named Picture1, on the form window Form1.

' This code is licensed according to the terms and conditions listed here.
' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Windows API Guide: ReleaseCapture Function

Public Declare Function SetCapture Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function ReleaseCapture Lib "user32.dll" () As Long

' Whenever the mouse moves, draw a line connecting the cursor's hot spot
to the center of Picture1. Of course, the line will be clipped withing the
boundaries
' of the picture box. To do this, Picture1 captures the mouse input when the form
loads, and releases it when the user clicks the mouse. For simplicity, the
' picture box's methods are used for drawing the line instead of using the
' proper API functions.

Private Sub Form1_Load()
' Have Picture1 capture mouse input. Also make sure that
' Picture1's scale mode is set to "Pixel".
Dim retval As Long  ' return value

retval = SetCapture(Picture1.hWnd)
Picture1.ScaleMode = 3

End Sub

Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
' Erase the previous line and draw a line connecting Picture1's center
to the mouse cursor. The line will be clipped at the boundary of the
picture box.
Static oldX As Long, oldY As Long  ' the previous mouse coordinates

' Erase the old line by drawing over it in the background color.
Picture1.Line (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2)-(oldX, oldY), _
       Picture1.BackColor
' Now draw the new line.
Picture1.Line (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2)-(X, Y)
' Save the mouse coordinates -- they'll be needed next time.
oldX = X: oldY = Y

End Sub

Private Sub Picture1_Click()
' When the mouse is clicked, release the mouse capture.
Dim retval As Long  ' return value

retval = ReleaseCapture()

End Sub

See Also

GetCapture, SetCapture

Category

Mouse
ReleaseDC Function

Declare Function ReleaseDC Lib "user32.dll" (ByVal hWnd As Long, ByVal hdc As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

ReleaseDC frees up the resources used when you use GetDC to get an object's device context. This function should not be used to destroy a device context obtained from CreateDC -- for those, use DeleteDC instead. This should be done after your program finishes using the device context. The function returns 0 if an error occurred or a 1 if successful.

hWnd

The handle of the object to free the resources of.

hdc

The device context of the object to free the resources of.

Example:

' Get the device context of the desktop window. This example doesn't ' use the DC for anything, but it could be used to copy the desktop image to another window.
Dim deskhwnd As Long   ' receives handle to the desktop window
Dim deskhdc As Long   ' receives device context of the desktop window
Dim retval As Long    ' return value

' Figure out the desktop's device context
deskhwnd = GetDesktopWindow()   ' get the desktop's handle
deskhdc = GetDC(deskhwnd)   ' get its device context

' deskhdc could be used here to do any number of things....

' Release the device context to free up resources
retval = ReleaseDC(deskhwnd, deskhdc)

See Also: DeleteDC, GetDC

Category: Devices
RemoveDirectory Function

Declare Function RemoveDirectory Lib "kernel32.dll" Alias "RemoveDirectoryA"
(ByVal lpPathName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

RemoveDirectory deletes a directory from a disk. The function will not delete any files or subdirectories inside
the directory. If the directory to delete is not completely empty, the function will fail. The function returns 1 if
successful, or 0 if an error occurred.

lpPathName
The directory to delete. The directory must be completely empty.

Example:

' Delete the directory C:\MyPrograms\TempData.
Dim retval As Long  ' return value

retval = RemoveDirectory("C:\MyPrograms\TempData")  ' delete the directory
If retval = 1 Then  ' success
    Debug.Print "C:\MyPrograms\TempData was successfully deleted."
Else
    Debug.Print "Deletion failed.  Make sure C:\MyPrograms\TempData is empty."
End If

See Also: CreateDirectory
Category: Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
RemoveMenu Function

Declare Function RemoveMenu Lib "user32.dll" (ByVal hMenu As Long, ByVal uPosition As Long, ByVal uFlags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RemoveMenu removes an item from a menu. If the item being removed is a submenu, the submenu is not actually destroyed by this function. Instead, the submenu is simply removed from the menu, allowing your program to use the submenu elsewhere.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hMenu
A handle to the menu to remove an item from.

uPosition
Either the menu item identifier of the item to remove, or the zero-based position of the item to remove, depending on the value of uFlags.

uFlags
One of the following flags specifying the type of information passed as uPosition:

- MF_BYCOMMAND
- MF_BYPOSITION
RemoveMenu Function

$uPosition$ is the menu item identifier of the item to remove.

MF_BYPOSITION
$uPosition$ is the zero-based index of the position of the item to remove.

Constant Definitions

Const MF_BYCOMMAND = &H0
Const MF_BYPOSITION = &H400

Example

Remove the "Maximize" and "Minimize" options from a form window's system menu. Note that doing this will not remove the maximize and minimize buttons on the window's title bar, although the buttons will stop working. In a real program, if you wanted to remove the minimize and maximize buttons from a window, the best way would be to do so before creating the window. To use this example, place a command button named cmdExample on a form window.

`This code is licensed according to the terms and conditions listed here.`

Declarations and such needed for the example:

`(Copy them to the (declarations) section of a module.)`

Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Public Declare Function RemoveMenu Lib "user32.dll" (ByVal hMenu As Long, ByVal uPosition As Long, ByVal uFlags As Long) As Long
Public Const MF_BYCOMMAND = &H0
Public Const SC_MINIMIZE = &HF020
Public Const SC_MAXIMIZE = &HF030

`*** Place the following code inside the form window. ***`

Private Sub cmdExample_Click()
  Dim hSysMenu As Long  ' handle to the window's system menu
  Dim retval As Long   ' return value of functions

  ' Get a handle to the window's system menu.
  hSysMenu = GetSystemMenu(Me.hWnd, 0)

  ' Remove the minimize and maximize options, using their item IDs.
  retval = RemoveMenu(hSysMenu, SC_MINIMIZE, MF_BYCOMMAND)
  retval = RemoveMenu(hSysMenu, SC_MAXIMIZE, MF_BYCOMMAND)
End Sub

See Also

InsertMenuItem
Category

Menus

Back to the Function list.
Back to the Reference section.
RemoveProp Function

Declare Function RemoveProp Lib "user32.dll" Alias "RemovePropA" (ByVal hWnd As Long, ByVal lpString As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

RemoveProp removes a window property from a window. This property must have been previously set using the SetProp function. Your program must remove all window properties it creates before the window they are attached to closes. However, never remove any window properties not created by your program. Although this function deletes the property itself, it does not affect the data referenced by the property in any way.

Return Value

If an error occurred, the function returns 0. If successful, the function returns a handle to the data which was attached to the property. The program is then responsible for deallocating this data, if necessary.

Visual Basic-Specific Issues

None.

Parameters

*hWnd*

A handle to remove one of the window properties of.
**lpString**

The name of the window property to remove.

### Example

```
' This code is licensed according to the terms and conditions listed here.

' Set the "LookupFile" property of window Form1 to a string.
' This example also shows how to remove the property.
Dim hStr As Long, pStr As Long  ' handle and pointer to the string
Dim thevalue As String  ' the string referenced by the handle
Dim retval As Long  ' return value

' Set the value of the string (this could be anything, really).
thevalue = "C:\Icons\default.ico"
' Create a memory block...
hStr = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(thevalue))
' ...and copy the string into it.
pStr = GlobalLock(hStr)  ' get a pointer to the block
retval = lstrcpy(pStr, text)  ' copy the string
retval = GlobalUnlock(hStr)  ' release the pointer

' The handle hStr now refers to a memory block holding the string. Set
' the "LookupFile" property to this memory block.
retval = SetProp(Form1.hWnd, "LookupFile", hStr)
' Note how we cannot yet free the memory block since it is still in use.

' *** INSERT OTHER CODE (such as the GetProp example) HERE ***

' The following code releases the "LookupFile" property and frees
' the memory block to which it points.
' (this code assumes the same Dims as above)
hStr = RemoveProp(Form1.hWnd, "LookupFile")
' The property is gone; now free the memory block.
retval = GlobalFree(hStr)
```

### See Also

[SetProp](#)

### Category

RestartDialog Function

Declare Function RestartDialog Lib "shell32.dll" Alias "#59" (ByVal hwndOwner As Long, ByVal lpstrReason As String, ByVal uFlags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Unknown.

The RestartDialog function is officially undocumented.

Description & Usage

RestartDialog prompts the user with a yes-or-no dialog box asking to either reboot, shut down, or log off the system. This dialog box is typically displayed at the end of an install routine, when a reboot is necessary to load altered system settings. If the user clicks the "Yes" button of the dialog, the reboot/shut down/log off operation is automatically begun.

Windows NT, 2000: All strings used by this function must be Unicode. Therefore, any strings passed to the function must first be converted into Unicode. Likewise, any strings output by the function also must be converted from Unicode into ANSI.

Return Value

If successful, the function returns one of the following values. However, if an error occurred, the function will return IDYES anyway. Therefore, the return value is an unreliable method of determining what the user selected.

IDNO
  The user selected "No".
IDYES
  The user selected "Yes".

Visual Basic-Specific Issues

None.

Parameters
hwndOwner

A handle to the window that is requesting the dialog box.

lpstrReason

Text to display inside the dialog box explaining why the reboot, shut down, or log off is necessary. Windows will append additional text to this message, so this string should end in one or two carriage return/line feed pairs (the intrinsic vbCrLf constant) or at least a space.

uFlags

One of the following flags specifying which operation to prompt the user about:

EWX_LOGOFF

Prompt the user to log off the system. The text appended to lpstrReason is as follows: "You must restart your computer before the new settings will take effect. (new line) Do you want to restart your computer now?"

EWX_REBOOT

Prompt the user to reboot the system. The text appended to lpstrReason is as follows: "You must restart your computer before the new settings will take effect. (new line) Do you want to restart your computer now?"

EWX_SHUTDOWN

Prompt the user to shut the system down. The text appended to lpstrReason is as follows: "Do you want to shut down now?"

Constant Definitions

Const IDNO = 7
Const IDYES = 6
Const EWX_LOGOFF = &H0
Const EWX_SHUTDOWN = &H1
Const EWX_REBOOT = &H2

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function RestartDialog Lib "shell32.dll" Alias "#59" (ByVal hwndOwner As Long, ByVal lpstrReason As String, ByVal uFlags As Long) As Long
Public Const EWX_REBOOT = &H2
Public Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type
Public Const VER_PLATFORM_WIN32_NT = 2
Public Const VER_PLATFORM_WIN32_WINDOWS = 1
Public Declare Function GetVersionEx Lib "kernel32.dll" Alias "GetVersionExA" (lpVersionInformation As OSVERSIONINFO) As Long

Windows API Guide: RestartDialog Function

' Prompt the user to reboot the computer. If he says "Yes", the function will
' automatically reboot the system. There's nothing much to it!
Private Sub Command1_Click()
    Dim s As String  ' the string to display in the dialog
    Dim ovi As OSVERSIONINFO  ' version of the OS
    Dim retval As Long  ' return value

    ' Display the following message in the dialog, in front of the text that
    ' Windows will automatically place there.
    s = "The install routine has made some changes to your computer. "

    ' If this is Windows NT or 2000, convert the string to Unicode.
    ovi.dwOSVersionInfoSize = Len(ovi)
    retval = GetVersionEx(ovi)
    If ovi.dwPlatformId = VER_PLATFORM_WIN32_NT Then
        s = StrConv(s, vbUnicode)
    End If

    ' Prompt the user.
    retval = RestartDialog(Form1.hWnd, s, EWX_RESTART)
End Sub

See Also

ExitWindowsDialog

Category

Shell

Back to the Function list.
Back to the Reference section.

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/r/restartdialog.html
RoundRect Function

Declare Function RoundRect Lib "gdi32.dll" (ByVal hdc As Long, ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

RoundRect draws a rectangle with rounded corners on a device. The rounded rectangle is drawn in the device's current and is filled using its current brush. The first two (x,y) coordinate pairs specified are the upper-left and lower-right corners of a comparable square-cornered rectangle. The third pair specifies the width and height of the rounded corner to use.

Return Value

If an error occurs, the function returns 0 (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- hdc
  A handle to the device context of the device to draw on.
- nLeftRect
  The x-coordinate of the upper-left corner of the corresponding square-edged rectangle.
- nTopRect
  The y-coordinate of the upper-left corner of the corresponding square-edged rectangle.
- nRightRect
  The x-coordinate of the lower-right corner of the corresponding square-edged rectangle.
- nBottomRect
  The y-coordinate of the lower-right corner of the corresponding square-edged rectangle.
The y-coordinate of the lower-right corner of the corresponding square-edged rectangle.

\( nWidth \)

The width of each rounded corner.

\( nHeight \)

The height of each rounded corner.

### Example

```vba
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function RoundRect Lib "gdi32.dll" (ByVal hdc As Long, ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long, ByVal nWidth As Long, ByVal nHeight As Long) As Long
Public Declare Function SelectObject Lib "gdi32.dll" (ByVal hdc As Long, ByVal hObject As Long) As Long
Public Declare Function GetStockObject Lib "gdi32.dll" (ByVal nIndex As Long) As Long
Public Const WHITE_PEN = 6
Public Const LTGRAY_BRUSH = 1

' Draw a white rounded rectangle on window Form1 with an upper-left
corners of (25,30) and a lower-right corner of (100,50). Give the rounded
corners a width of 10 and a height of 5. Fill the shape with light gray.
Dim hPen As Long ' handle to the pen to use to draw the shape
Dim hBrush As Long ' handle to the brush to use to draw the shape
Dim hOldPen As Long ' handle to Form1's previous pen
Dim hOldBrush As Long ' handle to Form1's previous brush
Dim retval As Long ' return value

' Get handles to the appropriate stock pen and brush.
hPen = GetStockObject(WHITE_PEN)
hBrush = GetStockObject(LTGRAY_BRUSH)
' Select the pen and brush for use by Form1.
hOldPen = SelectObject(Form1.hDC, hPen)
hOldBrush = SelectObject(Form1.hDC, hBrush)
' Draw the rounded rectangle.
retval = RoundRect(Form1.hDC, 25, 30, 100, 50, 10, 5)
' Restore Form1's previous pen and brush selections.
retval = SelectObject(Form1.hDC, hOldPen)
retval = SelectObject(Form1.hDC, hOldBrush)
```

### See Also

[Rectangle](http://216.26.168.92/vbapi/ref/r/roundrect.html)
Category

Filled Shapes

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SelectObject Function

Declare Function SelectObject Lib "gdi32.dll" (ByVal hdc As Long, ByVal hObject As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SelectObject selects a given object for use on a device. Possible objects to use with this function include bitmaps, brushes, fonts, pens, and regions. Once selected, this object will be used by the device whenever necessary. (For example, the selected brush will be used whenever the device needs to perform a fill.) The function returns a handle to the object previously selected by the device to do that task (e.g., the old brush). The program should re-select the old object when it is finished using it in order to preserve the device's default objects (see the example for a demonstration of this).

\textit{hdc}  
A device context to the device to select an object for.

\textit{hObject}  
A handle to the bitmap, brush, font, pen, or region to select for the device.

Example:

' Draw a rectangle with corners (10,20) and (175,100)  
' on window Form1. Use a solid yellow brush to fill the rectangle.
Dim hbrush As Long ' receives handle to the solid yellow brush
Dim holdbrush As Long ' receives handle to Form1's default brush
Dim retval As Long ' return value

hbrush = CreateSolidBrush(RGB(255, 255, 0)) ' create a solid yellow brush
' Save Form1's default brush so we can restore it after the program is finished
holdbrush = SelectObject(Form1.hDC, hbrush) ' select the brush
' Draw the rectangle filled using the solid yellow brush
retval = Rectangle(Form1.hDC, 10, 20, 175, 100)
' Restore Form1's previous brush before destroying the created one
retval = SelectObject(Form1.hDC, holdbrush) ' select old brush
retval = DeleteObject(hbrush) ' destroy the solid yellow brush

Category: Devices

Go back to the alphabetical Function listing.
Go back to the Reference section index.
send Function

Declare Function send Lib "wsock32.dll" (ByVal s As Long, buf As Any, ByVal length As Long, ByVal flags As Long) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The `send` function sends data to a network host via a connected socket. The socket must have already been connected to another computer via `connect`. The Winsock implementation automatically assembles the data into whatever packets, datagrams, or other units that are transported via the protocol being used.

Return Value

If successful, the function returns the number of bytes successfully sent through the socket (although the network host may not necessarily have received them). If an error occurred, the function returns SOCKET_ERROR (use `WSAGetLastError` to get the error code).

Visual Basic-Specific Issues

If specifying a string for `buf`, the ByVal keyword must be placed in front of it. See the example for a demonstration.

Parameters

- **s**: A descriptor of the connected socket to send the data through.
- **buf**: The data to send to the network host.
- **length**: The size in bytes of the data passed as `buf`.
- **flags**: A combination of the following flags specifying additional options:
  - MSG_DONTROUTE
Do not subject the data to routing. The Winsock implementation may choose to ignore this option if specified.

**MSG_OOB**
Send out-of-band data only.

## Constant Definitions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Const SOCKET_ERROR</td>
<td>-1</td>
</tr>
<tr>
<td>Const MSG_DONTROUTE</td>
<td>&amp;H4</td>
</tr>
<tr>
<td>Const MSG_OOB</td>
<td>&amp;H1</td>
</tr>
</tbody>
</table>

## Example

Download the main page of this web site ([http://www.vbapi.com](http://www.vbapi.com)). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and **WSACleanup** must be called at the end no matter what happens, the GoTo skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

```
' This code is licensed according to the terms and conditions listed [here](http://www.vbapi.com).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type **WSADATA**
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function **WSAStartup** Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As **WSADATA**) As Long
Public Declare Function **WSACleanup** Lib "wsock32.dll" () As Long
Public Type **HOSTENT**
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function **gethostbyname** Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function **htons** Lib "wsock32.dll" (ByVal hostshort As Integer) As Long
```
Windows API Guide: send Function

Integer Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long
Public Const SOCK_STREAM = 1
Public Type SOCKADDR
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type
Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, ByVal name As SOCKADDR, ByVal namelen As Long) As Long
Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, ByVal buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, ByVal buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

' A useful API macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA ' info about Winsock
    Dim sock As Long ' descriptor of the socket to use
    Dim retval As Long ' generic return value
    Dim pHostinfo As Long ' pointer to info about the server
    Dim hostinfo As HOSTENT ' info about the server
    Dim pIPAddress As Long ' pointer to server's IP address
    Dim ipAddress As Long ' server's IP address
    Dim sockinfo As SOCKADDR ' info about the socket
    Dim buffer As String ' communications buffer
    Dim reply As String ' reply from server
    Dim bytesleft As Long ' number of bytes left to read of response body
    Dim headerline As String ' line of HTTP response header

    ' Initialize a Winsock session.
    retval = WSAStartup(MAKEWORD(2, 2), wsockinfo)
    If retval <> 0 Then
        Debug.Print "Unable to initialize Winsock!"
    End If

    ' *** Place the following code inside the form window. ***
Exit Sub
End If

' Get the IP address of the server to connect to.
pHostinfo = gethostbyname("www.vbapi.com")
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve host!"
    GoTo Cleanup
End If

CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
If hostinfo.h_addrtype <> AF_INET Then
    Debug.Print "Couldn't get IP address of www.vbapi.com!"
    GoTo Cleanup
End If

CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4

' Create a socket to use for the TCP/IP connection.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = &HFFFFFFFF Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If

' Make a connection to the server.
With sockinfo
    ' Use the IP protocol family.
    .sin_family = AF_INET
    ' Connect to port 80 (the typical HTTP port).
    .sin_port = htons(80)
    ' IP address of the server to connect to.
    .sin_addr = ipAddress
    ' Dummy data that isn't used.
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If

' Send an HTTP request to GET the document /index.html.
buffer = "GET / HTTP/1.1" & vbCrLf & _
    "Host: www.vbapi.com" & vbCrLf & _
    "User-Agent: Winsock-Example-Program" & vbCrLf & vbCrLf
retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request. Waiting for reply...."

' Read from the socket until the entire HTTP response header is received.
' (i.e., until the connection times out or a double Cr-Lf pair is received)
reply = ""
buffer = Space(1024)  ' read in 1 KB chunks
Do
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    reply = reply & Left(buffer, retval)
Loop Until retval = 0 Or InStr(reply, vbCrLf & vbCrLf) <> 0
' Parse the header to see how many more bytes we need to read.
Do
    headerline = Left(reply, InStr(reply, vbCrLf) - 1)
    If LCase(Left(headerline, 16)) = "content-length: " Then
        bytesleft = Val(Right(headerline, Len(headerline) - 16))
    End If
    reply = Right(reply, Len(reply) - Len(headerline) - 2)
Loop While bytesleft = 0 Or Left(reply, 2) = vbCrLf
' Trim the rest of the header out of the reply.
reply = Right(reply, Len(reply) - InStr(reply, vbCrLf & vbCrLf) - 3)
bytesleft = bytesleft - Len(reply)
' Read the rest of the content of the response.
Do Until bytesleft = 0 Or retval = 0
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    reply = reply & Left(buffer, retval)
    bytesleft = bytesleft - retval
Loop
' Print the document that was received.
Debug.Print "Document Retrieved!"
Debug.Print
Debug.Print reply
Cleanup:
' Closes the socket, ends the Winsock session.
retval = closesocket(sock)
retval = WSACleanup()
End Sub

See Also
recv

Category
Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
SendInput Function

Declare Function SendInput Lib "user32.dll" (ByVal nInputs As Long, pInputs As INPUT_TYPE, ByVal cbSize As Long) As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 with Service Pack 3 (SP3) or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

SendInput synthesizes a series of keyboard, mouse, or other hardware inputs and adds them the input stream. The events generated by the function are not interspersed with any other input messages, user-created or otherwise.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the number of input events which were successfully added to the input stream.

Visual Basic-Specific Issues

None.

Parameters

nInputs

The number of elements in the array passed as pInputs.

pInputs

An array holding information about each input event to insert into the input stream. Each element corresponds to a single input event.

cbSize

The size in bytes of a single INPUT_TYPE structure (not the total size of the array passed as pInputs).

Example
Synthesize the user typing the letter P followed by clicking the right mouse button. Note how the information for each individual event is placed in its associated structure before copying it into the input array. This example runs when the user clicks button Command1. So, to run this example, you must first place a command button named Command1 on a form window.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type MOUSEINPUT
    dx As Long
    dy As Long
    mouseData As Long
    dwFlags As Long
    time As Long
    dwExtraInfo As Long
End Type
Public Const MOUSEEVENTF_RIGHTDOWN = &H8
Public Const MOUSEEVENTF_RIGHTUP = &H10
Public Type KEYBDINPUT
    wVk As Integer
    wScan As Integer
    dwFlags As Long
    time As Long
    dwExtraInfo As Long
End Type
Public Const VK_P = &H50  ' using vbKeyP instead would also work
Public Const KEYEVENTF_KEYUP = &H2
Public Type INPUT_TYPE
    dwType As Long
    xi(0 To 23) As Byte
End Type
Public Const INPUT_KEYBOARD = 1
Public Const INPUT_MOUSE = 0
Public Declare Function SendInput Lib "user32.dll" (ByVal nInputs As Long, pInputs As INPUT_TYPE, ByVal cbSize As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)

' *** Place the following code inside the form window. ***
Private Sub Command1_Click()
    Dim inputevents(0 To 3) As INPUT_TYPE  ' holds information about each event
    Dim keyevent As KEYBDINPUT             ' temporarily hold keyboard input info
    Dim mouseevent As MOUSEINPUT           ' temporarily hold mouse input info

    ' Load the information needed to imitate pressing the P key.
    With keyevent
        .wVk = VK_P                          ' the P key
        .wScan = 0                            ' not needed
    End With

    ' *** Place the following code inside the form window. ***
.dwFlags = 0      ' press the key down
.time = 0        ' use the default
.dwExtraInfo = 0  ' not needed

End With
' Copy the structure into the input array's buffer.
inputevents(0).dwType = INPUT_KEYBOARD
CopyMemory inputevents(0).xi(0), keyevent, Len(keyevent)

' Do the same as above, but for releasing the P key.
With keyevent
  .wVk = VK_P      ' the P key
  .wScan = 0      ' not needed
  .dwFlags = KEYEVENTF_KEYUP   ' release the key
  .time = 0      ' use the default
  .dwExtraInfo = 0  ' not needed
End With
inputevents(1).dwType = INPUT_KEYBOARD
CopyMemory inputevents(1).xi(0), keyevent, Len(keyevent)

' Load the information needed to imitate pressing the right mouse button.
With mouseevent
  .dx = 0            ' no horizontal movement
  .dy = 0            ' no vertical movement
  .mouseData = 0     ' not needed
  .dwFlags = MOUSEEVENTF_RIGHTDOWN   ' right button down
  .time = 0          ' use the default
  .dwExtraInfo = 0   ' not needed
End With
' Copy the structure into the input array's buffer.
inputevents(2).dwType = INPUT_MOUSE
CopyMemory inputevents(2).xi(0), mouseevent, Len(mouseevent)

' Do the same as above, but for releasing the right mouse button.
With mouseevent
  mouseevent.dx = 0           ' no horizontal movement
  mouseevent.dy = 0           ' no vertical movement
  mouseevent.mouseData = 0    ' not needed
  mouseevent.dwFlags = MOUSEEVENTF_RIGHTUP   ' right button up
  mouseevent.time = 0         ' use the default
  mouseevent.dwExtraInfo = 0  ' not needed
End With
' Copy the structure into the input array's buffer.
inputevents(3).dwType = INPUT_MOUSE
CopyMemory inputevents(3).xi(0), mouseevent, Len(mouseevent)

' Now that all the information for the four input events has been placed
' into the array, finally send it into the input stream.
SendInput 4, inputevents(0), Len(inputevents(0))

End Sub

See Also
SendMessage Function

Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**SendMessage** sends a message to a window. Specifically, the function calls that window’s procedure to handle the message. This function does not return until the window has completed processing the message.

Return Value

The function returns the return value of the message which was sent.

Visual Basic-Specific Issues

Whenever passing a string or a Long integer as `wParam` or `lParam`, the ByVal keyword must appear in front of the parameter. ByVal is not needed if a structure or a pointer is being passed. Additionally, if a literal number (i.e., one that is not stored in a variable) is passed, it must be encased in the CLng() Visual Basic function to force its data type to Long. See the example for a demonstration of ByVal usage.

Parameters

- **hWnd**
  - A handle to the window to send the message to. If this is HWND_BROADCAST, the message is sent to all open top-level windows.
- **Msg**
  - The identifier of the message to send.
- **wParam**
  - Additional message-specific data.
- **lParam**
  - Additional message-specific data.
Constant Definitions

Const HWND_BROADCAST = &HFFFF

Example

' This code is licensed according to the terms and conditions listed here.

' Open the Browse for Folder dialog box and display both the display name and
' the actual name of the folder (if it is not a virtual folder). Although any
' folders under My Computer can be selected, have the directory
' C:\StartHere selected by default.

' *** Place the following code in a module. ***

' This function compensates for the fact that the AddressOf operator
' can only be used in a function call. It returns the parameter
' passed to it.
Public Function DummyFunc(ByVal param As Long) As Long
    DummyFunc = param
End Function

' This function is the callback function for the dialog box. It sets
' the selected folder to C:\StartHere when the box is initialized.
Public Function BrowseCallbackProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal lParam As Long, ByVal lpData As Long) As Long
    Dim pathstring As String  ' name of path to set by default
    Dim retval As Long  ' return value

    ' If the BFFM_INITIALIZED message is received, set the
    ' default selection to C:\StartHere.
    Select Case uMsg
        Case BFFM_INITIALIZED
            pathstring = "C:\StartHere"  ' the path to make the default selection
            ' Send a message to the dialog box telling it to select this path.
            ' Note the use of ByVal and the CLng function here.
            retval = SendMessage(hwnd, BFFM_SETSELECTION, ByVal CLng(1), ByVal pathstring)
    End Select
    BrowseCallbackProc = 0  ' the function should always return 0
End Function

' *** Place the following code where you want to open the ***
' *** Browse for Folder dialog box. ***
Dim bi As BROWSEINFO  ' structure passed to the function
Dim pidl As Long  ' PIDL to the user's selection
Dim phystpath As String  ' string used to temporarily hold the physical path
Dim retval As Long  ' return value
' Initialize the structure to be passed to the function.
bi.hwndOwner = Form1.hWnd  ' window Form1 is the owner of the dialog box
' Specify the My Computer virtual folder as the root
retval = SHGetSpecialFolderLocation(Form1.hWnd, CSIDL_DRIVES, bi.pidlRoot)
' Make room in the buffer to get the [virtual] folder's display name
bi.pszDisplayName = Space(260)
bi.lpszTitle = "Please choose a folder."  ' Message displayed to the user
bi.ulFlags = 0  ' no flags are needed here
' Identify the callback function to use for the dialog box.  Note
' how our DummyFunc is needed because AddressOf only works
' inside a function call.
bi.lpfn = DummyFunc(AddressOf BrowseCallbackProc)
bi.lParam = 0  ' the callback function here doesn't need this
bi.iImage = 0  ' this will be set by the function

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
    ' Remove the empty space from the display name variable.
    bi.pszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
    Debug.Print "The user selected: "; bi.pszDisplayName
    ' If the folder is not a virtual folder, display its physical location.
    physpath = Space(260)  ' make room in the buffer
    retval = SHGetPathFromIDList(pidl, physpath)
    If retval = 0 Then
        Debug.Print "Physical Location: (virtual folder)"
    Else
        ' Remove the empty space and display the result.
        physpath = Left(physpath, InStr(physpath, vbNullChar) - 1)
        Debug.Print "Physical Location: "; physpath
    End If
' Free the pidl returned by the function.
CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to
' identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

Category

Messages

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetActiveWindow Function

Declare Function SetActiveWindow Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetActiveWindow makes a given window the active window for the program, giving it the focus. This window only becomes the foreground window if the application which owns it is the currently active program. This function can only be used on windows which the program owns. This function should be used carefully, since the user normally does not expect the active window to change unexpectedly. The function returns 1 if successful, or 0 if an error occurred.

hwnd
A handle to the window to set as the active window.

Example:

' Make the window Form1 the active window for the program. Note that
' this function will not make the window the foreground window if the user is currently
' working with a separate program.
Dim retval As Long ' return value
retval = SetActiveWindow(Form1.hWnd) ' set Form1 as the application's active window

See Also: GetActiveWindow, SetForegroundWindow

Category: Windows
SetArcDirection Function

Declare Function SetArcDirection Lib "gdi32.dll" (ByVal hdc As Long, ByVal ArcDirection As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetArcDirection sets the direction that arcs are drawn in on a graphics-capable device. Arcs can be drawn either clockwise or counterclockwise from the starting point to the ending point. Although Win 95/98 implements this function, that platform ignores the setting specified and always draws arcs counterclockwise! The function returns 1 if successful, or 0 if an error occurred.

**hdc**

The device context of the device to set the arc-drawing direction of.

**ArcDirection**

Exactly one of the following flags specifying which direction to draw arcs in:

- AD_CLOCKWISE = 2
  - Draw arcs clockwise from the starting point to the ending point.
- AD_COUNTERCLOCKWISE = 1
  - Draw arcs counterclockwise from the starting point to the ending point.

**Example:**

' Draw the arc that forms the top half of an ellipse. The ellipse
' is centered at (100, 100), has a width of 200, and has a height of 100. The arc is
drawn
' in red on the window Form1.
Dim retval As Long ' return value
Form1.ForeColor = RGB(255, 0, 0) ' set the drawing color to red
retval = SetArcDirection(Form1.hDC, AD_COUNTERCLOCKWISE) ' draw the arc
clockwise
'The ellipse is determined by the bounding rectangle (0,50)-(200,150).
'The ray to (200, 100) is due right; the ray to (0, 100) is due left.
retval = Arc(Form1.hDC, 0, 50, 200, 150, 200, 100, 0, 100)

See Also: AngleArc, Arc, ArcTo, GetArcDirection

Category: Lines & Curves

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetBrushOrgEx Function

Declare Function SetBrushOrgEx Lib "gdi32.dll" (ByVal hdc As Long, ByVal nXOrg As Long, ByVal nYOrg As Long, lppt As POINT_TYPE) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetBrushOrgEx sets the origin point for using a brush on a given device. Note that this function only takes effect on the next brush the device selects -- the currently selected brush in unaffected! The brush origin point determines the offset of the 8x8 block used to fill in areas (the brush). For example, an origin point of (2,3) would shift the fill pattern 2 pixels to the right and 3 pixels downward. The old brush origin point is put into the variable passed as lppt. The function returns 1 if successful, or 0 if an error occured.

hdc
A device context to the device to set the brush origin point of.

nXOrg
The x-coordinate of the new brush origin point. This must be between 0 and 7 inclusive.

nYOrg
The y-coordinate of the new brush origin point. This must be between 0 and 7 inclusive.

lppt
Receives the former brush origin point.

Example:

' Fill in rectangle (10,20)-(200,150) on window Form1 with a double diagonal-cross hatch pattern: one in green, one in blue. After the first draw/paint, the brush
' origin is adjusted to make a nice overlay effect.
Dim hbrush As Long ' receives handle to the brushes the program creates
Dim holdbrush As Long ' receives handle to the device's default brush
Dim oldorg As POINT_TYPE ' receives original origin point
Dim xorg As POINT_TYPE ' throw-away value
Dim retval As Long ' return value

' ** First, draw the rectangle using the green hatched pattern **
' Create the green hatched brush
hbrush = CreateHatchBrush(HS_DIAGCROSS, RGB(0, 255, 0))
' Set the brush origin point to (0,0) and store the old value (we'll restore it later)
retval = SetBrushOrgEx(Form1.hDC, 0, 0, oldorg)
' Select the hatched brush and create the rectangle
holdbrush = SelectObject(Form1.hDC, hbrush) ' select the brush

retval = **Rectangle**(Form1.hDC, 10, 20, 200, 150)  ' make the rectangle
' Restore the default brush temporarily and destroy the hatched brush
retval = **SelectObject**(Form1.hDC, holdbrush)  ' restore old brush
retval = **DeleteObject**(hbrush)  ' delete the hatched brush

' ** Now, redraw the rectangle using an offset blue hatched pattern **
' Create the blue hatched brush
hbrush = **CreateHatchBrush**(HS_DIAGCROSS, RGB(0, 0, 255))
' Set the brush origin point to (4,4) and ignore the old value
retval = **SetBrushOrgEx**(Form1.hDC, 0, 0, xorg)  ' we don't care about xorg
' Select the hatched brush and recreate the rectangle
holdbrush = **SelectObject**(Form1.hDC, hbrush)  ' select the brush
retval = **Rectangle**(Form1.hDC, 10, 20, 200, 150)  ' make the rectangle
' Restore the default brush and destroy the hatched brush
retval = **SelectObject**(Form1.hDC, holdbrush)  ' restore old brush
retval = **DeleteObject**(hbrush)  ' delete the hatched brush
' Now, restore the device's old brush origin point
retval = **SetBrushOrgEx**(Form1.hDC, oldorg.x, oldorg.y, xorg)

See Also: **GetBrushOrgEx**
Category: **Brushes**

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetCapture Function

Declare Function SetCapture Lib "user32.dll" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetCapture captures the mouse for the specified window. When a window captures the mouse, it receives all mouse input messages, including those which would otherwise be sent to other windows. Capturing the mouse input allows a window to keep track of all mouse actions via mouse messages, but no other windows will receive the mouse input. When the capture is no longer needed, the mouse capture should be ended by calling ReleaseCapture.

Return Value

If successful, the function returns a handle to the window that had previously captured the mouse, or zero if no window had captured the mouse. If an error occurred, the function also returns 0.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to capture the mouse input.

Example

The following example assumes that there is a picture box control, named Picture1, on the form window Form1.

' This code is licensed according to the terms and conditions listed here.
Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SetCapture Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function ReleaseCapture Lib "user32.dll" () As Long

Whenever the mouse moves, draw a line connecting the cursor's hot spot
to the center of Picture1. Of course, the line will be clipped within the boundaries
of the picture box. To do this, Picture1 captures the mouse input when the form loads, and releases it when the user clicks the mouse. For simplicity, the picture box's methods are used for drawing the line instead of using the proper API functions.

Private Sub Form1_Load()
    ' Have Picture1 capture mouse input. Also make sure that
    ' Picture1's scale mode is set to "Pixel".
    Dim retval As Long ' return value
    retval = SetCapture(Picture1.hWnd)
    Picture1.ScaleMode = 3
End Sub

Private Sub Picture1_MouseMove(Button As Integer, Shift As Integer, X As Single, Y As Single)
    ' Erase the previous line and draw a line connecting Picture1's center
to the mouse cursor. The line will be clipped at the boundary of the picture box.
    Static oldX As Long, oldY As Long ' the previous mouse coordinates
    ' Erase the old line by drawing over it in the background color.
    Picture1.Line (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2)-(oldX, oldY), _
        Picture1.BackColor
    ' Now draw the new line.
    Picture1.Line (Picture1.ScaleWidth / 2, Picture1.ScaleHeight / 2)-(X, Y)
    ' Save the mouse coordinates -- they'll be needed next time.
    oldX = X: oldY = Y
End Sub

Private Sub Picture1_Click()
    ' When the mouse is clicked, release the mouse capture.
    Dim retval As Long ' return value
    retval = ReleaseCapture()
End Sub

See Also

GetCapture, ReleaseCapture

Category

Mouse

Back to the Function list.
Back to the Reference section.

**Last Modified:** May 21, 2000
This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information] Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/s/setcapture.html](http://www.vbapi.com/ref/s/setcapture.html)
SetClassLong Function

Declare Function SetClassLong Lib "user32.dll" Alias "SetClassLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetClassLong sets a single 32-bit value from the information about the window class to which the specified window belongs. This function can also set a 32-bit value within the extra memory area associated with the window class.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the previous setting of the 32-bit value which was changed.

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to a window which belongs to the class to set a property of.

nIndex
To set a 32-bit value within the class's extra memory, set this to the zero-based offset of the first byte in the 32-bit block to set. Valid values are 0 to the number of bytes of extra memory minus 4, inclusive. To set a 32-bit property of the class, set this to one of the following flags specifying which 32-bit value to set from the window class:
  GCL_CBCLSEXTRA
  Set the size in bytes of the extra memory associated with the window class.
  dwNewLong
  is the new size of the block.
  GCL_CBWNDEXTRA
Set the size in bytes of the extra memory associated with each window belonging to the window class. *dwNewLong* is the new size of the block.

**GCL_HBRBACKGROUND**
Set the *brush* used to paint the backgrounds of windows belonging to the class. *dwNewLong* is a handle to the brush to use.

**GCL_HCURSOR**
Set the cursor associated with the class. *dwNewLong* is a handle to the cursor to use.

**GCL_HICON**
Set the icon associated with the class. *dwNewLong* is a handle to the icon to use.

**GCL_HICONSM**
Set the small icon associated with the class. *dwNewLong* is a handle to the small icon to use.

**GCL_HMODULE**
Change the module which registered the class. *dwNewLong* is a handle to the module to identify as the creator of the class.

**GCL_MENUNAME**
Set the string identifying the name of the menu resource associated with the class. *dwNewLong* is a *pointer* to the new string.

**GCL_STYLE**
Set the *window styles* associated with the class. *dwNewLong* is the new value.

**GCL_WNDPROC**
Set the *WindowProc* hook function to use as the window procedure for windows belonging to the window class. *dwNewLong* is a pointer to the function to use.

*dwNewLong*
The 32-bit value to set as something.

### Constant Definitions

```vbnet
Const GCL_CBCLSEXTRA = -20
Const GCL_CBWNDEXTRA = -18
Const GCL_HBRBACKGROUND = -10
Const GCL_HCURSOR = -12
Const GCL_HICON = -14
Const GCL_HMODULE = -16
Const GCL_MENUNAME = -8
Const GCL_STYLE = -26
Const GCL_WNDPROC = -24
```

### Example

```vbnet
' This code is licensed according to the terms and conditions listed here.

' Change the icon associated with whatever class window
' Form1 belongs to.
Dim hIcon As Long  ' handle to the new icon to use
Dim retval As Long  ' return value

' The icon to use is in C:\MyApp\newicon.ico.
hIcon = ExtractIcon(App.hInstance, "C:\MyApp\newicon.ico", 0)
' Set this icon as the new class icon.
```

retval = SetClassLong(Form1.hWnd, GWL_HICON, hIcon)  ' set the class's icon, not the window's
' Note that here we can't destroy the icon we extracted because it is still in use by the class.

See Also

GetClassLong, SetWindowLong

Category

Window Classes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 28, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setclasslong.html
SetCursor Function

Declare Function SetCursor Lib "user32.dll" (ByVal hCursor As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetCursor sets the image used to represent the mouse cursor. The new cursor can be any valid cursor that has either been created or loaded. If successful, the function returns a handle to the old cursor image. If unsuccessful, the function returns 0.

hCursor
The handle to the new cursor to use to represent the mouse pointer.

Example:

' Display the application starting (arrow and hourglass) Windows cursor for three seconds. The cursor resource is loaded from Windows. Then ' restore the old cursor (whatever it happens to be).
Dim hcursor As Long ' receives handle to application starting cursor
Dim holdcursor As Long ' receives handle to previously used cursor
Dim retval As Long ' throw-away return value

hcursor = LoadCursor(0, IDC_APPSTARTING) ' load Windows's application starting cursor
holdcursor = SetCursor(hcursor) ' set it to the new cursor
Sleep 3000 ' wait for 3 seconds
retval = SetCursor(holdcursor) ' set it to the previous cursor

See Also: GetCursor, SetSystemCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
Windows API Guide: SetCursor Function

This page is at http://www.vbapi.com/ref/s/setcursor.html
SetCursorPos Function

Declare Function SetCursorPos Lib "user32.dll" (ByVal x As Long, ByVal y As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetCursorPos sets the position of the mouse cursor. If you try to set the coordinates outside of the range of the display (for example, to (700,40) on a 640x480 display) or outside the confining rectangle (set by ClipCursor), the cursor will just go to the edge of the screen or the rectangle. The function returns 0 if an error occured, or 1 if successful.

\[x\]
\[y\]

The x coordinate to move the cursor to.

The y coordinate to move the cursor to.

Example:

' Move the mouse cursor to the point (100,200) on the screen
Dim retval As Long  ' return value
retval = SetCursorPos(100, 200)  ' move the cursor to (100,200)

See Also: GetCursorPos
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetDoubleClickTime Function

Declare Function SetDoubleClickTime Lib "user32.dll" (ByVal wCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetDoubleClickTime sets the maximum amount of time allowed between successive mouse clicks for Windows to determine it as a double click. This function alters how Windows interprets a double click, so of course all applications as well as Windows itself will be affected. Be careful using this function, since the user usually sets the double click speed via Windows's Control Panel, so he/she may not expect a change in the double click speed. The function returns 1 if successful, or 0 if an error occurred.

wCount
The maximum amount of time, in milliseconds, to allow between successive clicks for Windows to interpret it as a double click. A value of 0 restores Windows's default double click speed of 500 milliseconds.

Example:

' Set the maximum double click speed to 1 second.
Dim retval As Long ' return value

retval = SetDoubleClickTime(1000) ' set the double click speed to 1 second
Debug.Print "The double click speed is now 1 second (1000 milliseconds)."

See Also: GetDoubleClickTime
Category: Mouse
SetEnvironmentVariable Function

Declare Function SetEnvironmentVariable Lib "kernel32.dll" Alias "SetEnvironmentVariableA" (ByVal lpName As String, ByVal lpValue As String) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

SetEnvironmentVariable sets the value of one of the computer's environment variables.

Return Value

If successful, the function returns a nonzero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- **lpName**
  
  The name of the environment variable to set.

- **lpValue**
  
  The value to give to the environment variable.
Example

Change the value of the TEMP and TMP environment variables, which refer to the directory used to store temporary files, to "D:\Temp". This probably isn't the best way to try to change the temporary file directory, but remember, this is just an example. The example runs when the user clicks button Command1. Naturally, to use this example, you must place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SetEnvironmentVariable Lib "kernel32.dll" Alias "SetEnvironmentVariableA" (ByVal _
    lpName As String, ByVal lpValue As String) As Long

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim retval As Long  ' return value

    ' Change the value of the TEMP and TMP environment variables.
    retval = SetEnvironmentVariable("TEMP", "D:\Temp")
    retval = SetEnvironmentVariable("TMP", "D:\Temp")
    ' That's all there is to it!

End Sub

See Also

GetEnvironmentVariable

Category

Processes & Threads

Back to the Function list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
SetFileAttributes Function

Declare Function SetFileAttributes Lib "kernel32.dll" Alias "SetFileAttributesA" (ByVal lpFileName As String, ByVal dwFileAttributes As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetFileAttributes changes the attributes of a file or a directory. The four attributes you can set are archive, read-only, hidden, and system status -- the other attributes merely reflect unchangeable properties of the file. Any of the four can be on or off in any order. The function returns 0 if an error occurred, or 1 if successful.

**lpFileName**
The filename or directory, including the full path, to change the attributes of.

**dwFileAttributes**
One or more of the following flags specifying the attributes to set (those that can only be set by the operating system are not listed here):

- `FILE_ATTRIBUTE_ARCHIVE = &H20`
  An archive file (which most files are).
- `FILE_ATTRIBUTE_HIDDEN = &H2`
  A hidden file, not normally visible to the user.
- `FILE_ATTRIBUTE_NORMAL = &H80`
  An attribute-less file (cannot be combined with other attributes).
- `FILE_ATTRIBUTE_READONLY = &H1`
  A read-only file.
- `FILE_ATTRIBUTE_SYSTEM = &H4`
  A system file, used exclusively by the operating system.

**Example:**

```vba
' Set the file C:\MyProgram\secret.dat to a hidden, read-only, ' archive file.
Dim fileattrs As Long  ' file attributes
Dim retval As Long  ' return value

fileattrs = FILE_ATTRIBUTE_ARCHIVE Or FILE_ATTRIBUTE_HIDDEN Or FILE_ATTRIBUTE_READONLY
retval = SetFileAttributes("C:\MyProgram\secret.dat", fileattrs)  ' set the file's attributes
```

**See Also:** GetFileAttributes

**Category:** Files
SetFilePointer Function

Declare Function SetFilePointer Lib "kernel32.dll" (ByVal hFile As Long, ByVal lDistanceToMove As Long, lpDistanceToMoveHigh As Long, ByVal dwMoveMethod As Long) As Long

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

*SetFilePointer* changes the position of the file pointer in an open file. The pointer can be moved either forwards or backwards relative to the beginning of the file, the current file pointer position, or the end of the file. The file pointer cannot be moved more than one byte past the last byte of the file (where it would point to the end of the file).

When calculating the value for *lDistanceToMove* and the value to set the variable passed as *lpDistanceToMoveHigh* for negative numbers (i.e., moving the file pointer backwards), you cannot simply take the negative of the absolute value. First put the absolute values into those parameters. For *lpDistanceToMoveHigh*, perform a bitwise NOT operation on the value. For *lDistanceToMove*, perform a bitwise NOT operation followed by adding 1. See the example for a demonstration of this procedure, which is necessary to produce a 64-bit negative number from a composite of the two dwords.

Return Value

If successful, the function returns the low-order dword of the new file pointer position. The high-order dword of the file pointer position is placed into the variable passed as *lpDistanceToMoveHigh*. If an error occurred, the function returns -1 (use *GetLastError* to get the error code).

Visual Basic-Specific Issues

None.

Parameters

*hFile*

A handle to the open file to move the file pointer of.
Windows API Guide: SetFilePointer Function

**lDistanceToMove**

The low-order dword of the 64-bit integer identifying the number of bytes to move the file pointer. For negative values, consult the second paragraph under Description & Usage above.

**lpDistanceToMoveHigh**

Variable that contains the high-order dword of the 64-bit integer identifying the number of bytes to move the file pointer. For negative values, consult the second paragraph under Description & Usage above. After the call, the variable receives the high-order dword of the new file pointer.

**dwMoveMethod**

One of the following flags specifying the reference point to move the file pointer from:

- **FILE_BEGIN**
  The beginning of the file; i.e., the very first byte of the file.

- **FILE_CURRENT**
  The current position of the file pointer.

- **FILE_END**
  The end of the file; i.e., immediately after the very last byte of the file.

**Constant Definitions**

Const FILE_BEGIN = 0
Const FILE_CURRENT = 1
Const FILE_END = 2

**Example**

Read two four-character strings from the file "C:\Test\myfile.txt": one string starting at the third byte from the beginning, and the other starting five bytes from the end. To use this example, place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreateFile Lib "kernel32.dll" Alias "CreateFileA" (ByVal lpFileName As String, _
  ByVal dwDesiredAccess As Long, ByVal dwShareMode As Long, _
  lpSecurityAttributes As Any, _,
  ByVal dwCreationDisposition As Long, ByVal dwFlagsAndAttributes As Long, _
  ByVal hTemplateFile As Long) As Long
Public Const GENERIC_READ = &H80000000
Public Const FILE_SHARE_READ = &H1
Public Const OPEN_EXISTING = 3
Public Const FILE_ATTRIBUTE_ARCHIVE = &H20
Public Declare Function SetFilePointer Lib "kernel32.dll" (ByVal hFile As Long, ByVal lDistanceToMove _,
  lpDistanceToMoveHigh As Long, ByVal dwMoveMethod As Long) As Long
Public Const FILE_BEGIN = 0
Public Const FILE_END = 2
Public Declare Function ReadFile Lib "kernel32.dll" (ByVal hFile As Long, ByVal lpBuffer As Any, _
  ByVal nNumberOfBytesToRead As Long, ByVal lpNumberOfBytesRead As Long, ByVal lpOverlapped As Long) As Long
Windows API Guide: SetFilePointer Function

As Any) As Long
Public Declare Function CloseHandle Lib "kernel32.dll" (ByVal hObject As Long) As Long

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim strbuf As String * 4 ' receives four-byte string from the file
    Dim lowbyte As Long ' low dword of file pointer position
    Dim highbyte As Long ' high dword of file pointer position
    Dim numread As Long ' receives number of bytes read from file
    Dim hFile As Long ' handle of the open file
    Dim retval As Long ' return value

    ' Open the file for read-level access, if it already exists.
    hFile = CreateFile("C:\Test\myfile.txt", GENERAL_READ, FILE_SHARE_READ, ByVal CLng(0), _
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
    If hFile = -1 Then
        Debug.Print "Unable to open the file -- it probably does not exist."
        Exit Sub
    End If

    ' Keep in mind that, although the lpDistanceToMoveHigh parameter
    ' must be passed a variable, the lDistanceToMove parameter can be
    ' passed either a variable or a literal constant. For clarity, the
    ' following code uses a variable for lDistanceToMove.

    ' Set the file pointer to 2 bytes after the beginning of the file -- i.e.,
    the third byte position.
    lowbyte = 2
    highbyte = 0
    lowbyte = SetFilePointer(hFile, lowbyte, highbyte, FILE_BEGIN)
    ' Now read four characters and display them.
    retval = ReadFile(hFile, ByVal strbuf, 4, numread, ByVal CLng(0))
    Debug.Print "Four-character string starting at byte 3: "; strbuf

    ' Set the file pointer to 5 bytes before the beginning of the file. Note how
    ' and highbyte numbers must be manipulated to represent a negative value.
    lowbyte = (Not 5) + 1
    highbyte = Not 0
    lowbyte = SetFilePointer(hFile, lowbyte, highbyte, FILE_END)
    ' Now read four characters from here and display them.
    retval = ReadFile(hFile, ByVal strbuf, 4, numread, ByVal CLng(0))
    Debug.Print "Four-character string starting at the fifth byte from the end: "; strbuf

    ' Close the file.
    retval = CloseHandle(hFile)
End Sub

See Also

ReadFile, WriteFile

Category

Files

Go back to the Function listing.
Go back to the Reference section index.

---

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setfilepointer.html
SetFileTime Function

Declare Function SetFileTime Lib "kernel32.dll" (ByVal hFile As Long, lpCreationTime As FILETIME, lpLastAccessTime As FILETIME, lpLastWriteTime As FILETIME) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

SetFileTime sets the creation, last-accessed, and last-modified (last written-to) dates and times associated with a file. All the times are in UTC time (Coordinated Universal Time, a.k.a. Greenwich Mean Time (GMT)), not in the system's local time. The times actually stored on the system may vary slightly from the times passed to the function because file times are not stored with perfect resolution (for example, seconds data may be ignored).

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters
**hFile**  
A [handle](https://docs.microsoft.com/en-us/windows/win32/fileio/handle-and-handles) to the opened file to set the times of. The file must have been opened with at least write-level access.

**lpCreationTime**  
The date and time to set as the file's creation time.

**lpLastAccessTime**  
The date and time to set as the file's last access time.

**lpLastWriteTime**  
The date and time to set as the file's last write-to (modification) time.

### Example

```vbnet
' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/s/setfiletime.html)

' Set the modification time of C:\MyApp\test.txt to
' the current system date and time. Leave the other times as they
' were before calling the function.
Dim hFile As Long ' handle to the opened file
Dim ctime As _FILETIME ' the time of creation
Dim atime As _FILETIME ' the time of last access
Dim mtime As _FILETIME ' the time of last modification
Dim retval As Long ' return value

' First, open the file C:\MyApp\test.txt for both read-level and
' write-level access, since we need to do both.
hFile = CreateFile("C:\MyApp\test.txt", GENERIC_READ Or GENERIC_WRITE,
FILE_SHARE_READ, ByVal CLng(0), OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hFile = -1 Then
    Debug.Print "Could not open the file successfully -- aborting."
    ' terminate the program
End If

' Next, get the creation, last-access, and last-modification times.
retval = GetFileTime(hFile, ctime, atime, mtime)

' Get the system time (already in UTC) as a _FILETIME structure.
GetSystemTimeAsFileTime mtime
' Set the retrieved creation and access times and the new modification
' time as the file's times.
retval = SetFileTime(hFile, ctime, atime, mtime)

' Close the file to free up resources.
retval = CloseHandle(hFile)
```

---

See Also

GetFileTime

Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetFocus Function

Declare Function SetFocusAPI Lib "user32.dll" Alias "SetFocus" (ByVal hWnd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetFocus gives a window the input focus. The window must be owned by the thread calling the function, however -- the function will not set the focus to another program's window.

Return Value

If successful, the function returns a handle to the window that previously had the input focus. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

This function behaves exactly like the SetFocus method of many VB controls. However, because this function shares its name with this method, it must be renamed to something like SetFocusAPI when you use it in a VB program, or else the compiler will give you an error. See the example for a demonstration of this.

Parameters

$hWnd$

A handle to the window to give the input focus.

Example

When the form loads, immediately give button cmdGetsFocus the keyboard focus. Although an easier way to do this is through the button's SetFocus method, the API function is useful if you only have a handle to the window you want to give...
the focus. To use this example, you must place a button named cmdGetsFocus on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SetFocusAPI Lib "user32.dll" Alias "SetFocus" (ByVal hWnd As Long) As Long

' *** Place the following code inside a form window. ***

Private Sub Form_Load()
    Dim retval As Long  ' return value
    ' Give the input focus to cmdGetsFocus right away.
    retval = SetFocusAPI(cmdGetsFocus hWnd)
End Sub

See Also

GetFocus

Category

Windows

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/(?/name).html
SetForegroundWindow Function

Declare Function SetForegroundWindow Lib "user32.dll" (ByVal hwnd As Long) As Long

Platforms: Win 95/98, Win NT

SetForegroundWindow makes the specified window the current foreground window and gives it the focus. This function should only be used with windows which your program owns. Of course this function should be used with caution, since the user usually doesn't expect the foreground window to change unexpectedly. The function tells Windows to somehow draw the user's attention to the window, such as by flashing its icon in the taskbar. The function returns 1 if successful, or 0 if an error occurred.

hwnd
A handle to the window to set as the foreground window.

Example:

' Make the window Form1 the current foreground window. The operating system will somehow draw the user to the window.
Dim retval As Long  ' return value

retval = SetForegroundWindow(Form1.hWnd)  ' set Form1 as the foreground window

See Also: GetForegroundWindow, SetActiveWindow

Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetKeyboardState Function

Declare Function SetKeyboardState Lib "user32.dll" (lpKeyState As Byte) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetKeyboardState sets the state of every key on the keyboard. Each element of the 256-element array identifies information about the virtual-key whose virtual-key code matches the index of the element. If the &H1 bit is set, that key is considered toggled. If the &H80 bit is set, the key is considered to be currently pressed down. The keyboard information set by this function is thread-specific; its settings do not necessarily change key states pertaining to the system as a whole.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpKeyState
A 256-element byte array which specifies the key status information for all virtual-keys. Each key is identified by the element corresponding with the key's virtual key code. Windows NT, 2000: In addition to the virtual keys, the array also sets information which distinguishes between the left and right Ctrl, Alt, and Shift keys, which are stored in the array at the following indices:

VK_LSHIFT
   The left Shift key.
VK_RSHIFT
   The right Shift key.
VK_LCONTROL
   The left Ctrl key.
VK_RCONTROL
   The right Ctrl key.
VK_LMENU
   The left Alt key.
VK_RMENU
   The right Alt key.

Constant Definitions

Const VK_LSHIFT = &HA0
Const VK_RSHIFT = &HA1
Const VK_LCONTROL = &HA2
Const VK_RCONTROL = &HA3
Const VK_LMENU = &HA4
Const VK_RMENU = &HA5

Example

' This code is licensed according to the terms and conditions listed here.

' Set the toggle status for every key on the keyboard to "not toggled." This change only applies to the current thread.
Dim keystates(0 To 255) As Byte ' holds states of entire keyboard
Dim c As Integer ' counter variable
Dim retval As Long ' return value

' First, get the current state of the keyboard.
retval = GetKeyboardState(keystates(0))

' Now, loop through each element and explicitly set the toggle bit to 0.
For c = 0 To 255
   ' Make sure the &H1 bit is not set.
   keystates(c) = keystates(c) And (Not &H1)
```
' Finally, set this to the current keyboard state.
retval = SetKeyboardState((keystates(0))
```

See Also

GetKeyboardState

Category

Keyboard

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 5, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000 Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setkeyboardstate.html
SetLastError Function

Declare Sub SetLastError Lib "kernel32.dll" (ByVal dwErrCode As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetLastError sets the API function error code for the calling thread. This error code is usually used to give a more detailed description of an error rather than a simple failure notification. The error code set by this function persists either until the next call to SetLastError or SetLastErrorEx or until another API function sets the error code from its operations (almost all of them call SetLastError internally).

Return Value

SetLastError does not return a value.

Visual Basic-Specific Issues

None.

Parameters

dwErrCode

The error code of the error to report. If you define your own error codes for use in your program, be sure that bit 29 (&H20000000) in it is set. Bit 29 denotes an application-defined error code; no error codes used by Windows have that bit set.
Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/s/setlasterror.html).

' Use **SetLastError** to test the failure condition for another API function. By setting the API error code, the error handling code can be run by simulating an actual error.

Dim retval As Long ' return value

' Get the attributes of the file C:\autoexec.bat.
retval = [GetFileAttributes]("C:\autoexec.bat")

' Now "fake" a File Not Found error by setting the returned value to zero and setting the API error code to the proper value.
' (Obviously, these two lines would be removed after testing the error handler routine.)
retval = 0 ' make it look like the function failed
**SetLastError** ERROR_FILE_NOT_FOUND

' Now try to trap for the error.
If retval = 0 Then ' some error occured
    Select Case Err.LastDllError ' see the [GetLastError](http://216.26.168.92/vbapi/ref/s/getlasterror.html) page for usage of this
        Case ERROR_FILE_NOT_FOUND ' File Not Found
            Debug.Print "The specified file could not be found."
        Case Else ' ???
            Debug.Print "An unknown and untrapped error occurred."
    End Select
Else
    Debug.Print "The function call was successful."
End If

See Also

[GetLastError], [SetLastErrorEx]

Category

[Errors]


SetLastErrorEx Function

Declare Sub SetLastErrorEx Lib "kernel32.dll" (ByVal dwErrCode As Long, ByVal dwType As Long)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

SetLastErrorEx sets the API function error code for the calling thread. This error code is usually used to give a more detailed description of an error rather than a simple failure notification. The error code set by this function persists either until the next call to SetLastError or SetLastErrorEx or until another API function sets the error code from its operations (almost all of them call SetLastError internally).

Return Value

SetLastError does not return a value.

Visual Basic-Specific Issues

None.

Parameters

* **dwErrCode**
  
  The [error code](http://216.26.168.92/vbapi/ref/s/setlasterrorex.html) of the error to report. If your define your own error codes for use in your program, be sure that bit 29 (&H20000000) in it is set. Bit 29 denotes an application-defined error code; no error codes used by Windows have that bit set.

* **dwType**
Reserved -- set to 0.

Example

' This code is licensed according to the terms and conditions listed here.

' Use SetLastErrorEx to test the failure condition for another API function. By setting the API error code, the error handling code can be run by simulating an actual error.
Dim retval As Long ' return value

' Get the attributes of the file C:\autoexec.bat.
retval = GetFileAttributes("C:\autoexec.bat")

' Now "fake" a File Not Found error by setting the returned value to zero and setting the API error code to the proper value.
' (Obviously, these two lines would be removed after testing ' the error handler routine.)
retval = 0 ' make it look like the function failed
SetLastErrorEx ERROR_FILE_NOT_FOUND

' Now try to trap for the error.
If retval = 0 Then ' some error occured
   Select Case Err.LastDllError ' see the GetLastError page for usage of this Case ERROR_FILE_NOT_FOUND ' File Not Found
      Debug.Print "The specified file could not be found."
   Case Else ' ???
      Debug.Print "An unknown and untrapped error occured."
   End Select
Else
   Debug.Print "The function call was successful."
End If

See Also

GetLastError, SetLastError

Category

Errors

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetMenuItemInfo Function

Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetMenuItemInfo changes the attributes of an existing menu item. This function can alter almost anything about the menu item, including whether it is checked, enabled, or default, among other things.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0 (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- **hMenu**
  A handle to the menu that contains the item to change.

- **uItem**
  Identifies the menu item to change. This could be either a position or a menu item identifier, depending on fByPosition.

- **fByPosition**
  If this is a non-zero value, uItem indicates the item by using its zero-based position. (For example, the first item in the menu has a position of 0.) If this is zero, then uItem is the unique menu item identifier of the item.

- **lpmii**
  Describes the changes to make to the menu item.
Example

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
'There's quite a few declarations for this example, but it's worth it!
Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long
Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long
Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1
Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd _
    lpOffset As Long, ByVal dwNewValue As Long) As Long
As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = -4
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116

' Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.
' Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

' *** Place the following code in a module. ***

Public pOldProc As Long ' pointer to Form1's previous window procedure
Public ontop As Boolean ' identifies if Form1 is always on top or not

' The following function acts as Form1's window procedure to process messages.
Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
Dim hSysMenu As Long ' handle to Form1's system menu
Dim mii As MENUITEMINFO ' menu item information for Always On Top
Dim retval As Long ' return value

Select Case uMsg
Case WM_INITMENU
' Before displaying the system menu, make sure that the Always On Top option is properly checked.
  hSysMenu = GetSystemMenu(hwnd, 0)
  With mii
    ' Size of the structure.
    .cbSize = Len(mii)
    ' Only use what needs to be changed.
    .fMask = MIIM_STATE
    ' If Form1 is now always on top, check the item.
    .fState = MFS_ENABLED Or IIf(ontop, MFS_CHECKED, 0)
  End With
  retval = SetMenuItemInfo(hSysMenu, 1, 0, mii)
  WindowProc = 0
Case WM_SYSCOMMAND
  ' If Always On Top (ID = 1) was selected, change the on top/not on top setting of Form1 to match.
  If wParam = 1 Then
    ' Reverse the setting and make it the current one.
    ontop = Not ontop
    retval = SetWindowPos(hwnd, IIf(ontop, HWND_TOPMOST, HWND_NOTOPMOST), _
                          0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
  End If
  WindowProc = 0
End Select
WindowProc = 0

Else
    ' Some other item was selected. Let the previous window procedure
    ' process it.
    WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End If

Case Else
    ' If this is some other message, let the previous procedure handle it.
    WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End Select

End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the
' new window procedure.
Private Sub Form_Load()
    Dim hSysMenu As Long     ' handle to the system menu
    Dim count As Long        ' the number of items initially on the menu
    Dim mii As MENUITEMINFO  ' describes a menu item to add
    Dim retval As Long       ' return value

    ' Get a handle to the system menu.
    hSysMenu = GetSystemMenu(Form1.hWnd, 0)
    ' See how many items are currently in it.
    count = GetMenuItemCount(hSysMenu)

    ' Add a separator bar and then Always On Top to the system menu.
    With mii
        ' The size of the structure.
        .cbSize = Len(mii)
        ' What parts of the structure to use.
        .fMask = MIIM_ID Or MIIM_TYPE
        ' This is a separator.
        .fType = MFT_SEPARATOR
        ' It has an ID of 0.
        .wID = 0
    End With

    ' Add the separator to the end of the system menu.
    retval = InsertMenuItem(hSysMenu, count, 1, mii)

    ' Likewise, add the Always On Top command.
    With mii
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        ' This is a regular text item.
        .fType = MFT_STRING
        ' The option is enabled.
        .fState = MFS_ENABLED
        ' It has an ID of 1 (this identifies it in the window procedure).
    End With

End Sub
.wID = 1
    ' The text to place in the menu item.
    .dwTypeData = ":&Always On Top"
    .cch = Len(.dwTypeData)
End With
    ' Add this to the bottom of the system menu.
    retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)

    ' Set the custom window procedure to process Form1's messages.
    ontop = False
    pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' Before unloading, restore the default system menu and remove the
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value

    ' Replace the previous window procedure to prevent crashing.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
    ' Remove the modifications made to the system menu.
    retval = GetSystemMenu(Form1.hWnd, 1)
End Sub

See Also

GetMenuItemInfo

Category

Menus

Back to the Function list.
Back to the Reference section.

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setmenuiteminfo.html
SetParent Function

Declare Function SetParent Lib "user32.dll" (ByVal hWndChild As Long, ByVal hWndNewParent As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetParent moves a window from having one parent window to another. If needed, the window itself moves so it can be "inside" its new parent. The child window can also become independent by making it a child of the desktop.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a handle to the child window's former parent window.

Visual Basic-Specific Issues

None.

Parameters

hWndChild
The handle of the window to change the parent of.

hWndNewParent
The handle of the window to become the new parent of the child window. To make the desktop the parent, pass 0 for this parameter.

Example

Move button Command1, which is a child window, from its old parent window of Form1 to its new parent window of Form2.

```
Dim oldhwnd As Long  ' receives handle of button's former parent
oldhwnd = SetParent(Command1.hWnd, Form2.hWnd)  ' button is now in window Form2.
```

See Also

IsChild, GetParent

Category

Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setparent.html
SetPixel Function

Declare Function SetPixel Lib "gdi32.dll" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long, ByVal crColor As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetPixel sets the color of a single pixel on a device. Because of color matching on the device, sometimes the color actually used on the device may differ from the color specified by the function. The function will fail if the pixel specified is not visible or out of range when the function is called.

Return Value

If an error occurred, the function returns -1 (call GetLastError to get the error code). If successful, the function returns the RGB value of the color which was actually used to color the pixel on the device.

Visual Basic-Specific Issues

None.

Parameters

\( hdc \)

A handle to a device context to the device to color a pixel of.

\( X \)
The x-coordinate of the pixel to color.

$Y$

The y-coordinate of the pixel to color.

$crColor$

The RGB value of the color to set the pixel to.

Example

' This code is licensed according to the terms and conditions listed here.

' Randomly color all the pixels in window Form1. This example
' gets the rectangle of Form1 and iterates through all the points
' (pixels) inside of it.
Dim winrect As RECT  ' rectangle of window Form1
Dim rgbval As Long  ' RGB value of the randomly selected color
Dim x As Long, y As Long  ' counters for x and y coordinates
Dim retval As Long  ' return value

' Get the rectangle of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)

' Loop through each pixel within Form1.
For y = 0 To winrect.bottom - winrect.top
    For x = 0 To winrect.right - winrect.left
        ' Select a random color by choosing a value between 0 and 255
        ' inclusive for each component of the color.
        rgbval = RGB(Int(256 * Rnd), Int(256 * Rnd), Int(256 * Rnd))
        ' Set the pixel to the color above.
        retval = SetPixel(Form1.hDC, x, y, rgbval)
    Next x
Next y

See Also

GetPixel, SetPixelV

Category

Bitmaps

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetPixelV Function

Declare Function SetPixelV Lib "gdi32.dll" (ByVal hdc As Long, ByVal X As Long, ByVal Y As Long, ByVal crColor As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetPixelV sets the color of a single pixel on a device. Because of color matching on the device, sometimes the color actually used on the device may differ from the color specified by the function. The function will fail if the pixel specified is not visible or out of range when the function is called. This function executes slightly more quickly than SetPixel because this function does not need to return the actual color used by the device.

Return Value

If an error occurred, the function returns 0 (call GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

hdc

A handle to a device context to the device to color a pixel of.
The x-coordinate of the pixel to color.

Y

The y-coordinate of the pixel to color.

crColor

The RGB value of the color to set the pixel to.

Example

' This code is licensed according to the terms and conditions listed here.

' Randomly color all the pixels in window Form1. This example ' gets the rectangle of Form1 and iterates through all the points ' (pixels) inside of it.
Dim winrect As RECT  ' rectangle of window Form1
Dim rgbval As Long  ' RGB value of the randomly selected color
Dim x As Long, y As Long  ' counters for x and y coordinates
Dim retval As Long  ' return value

' Get the rectangle of window Form1.
retval = GetClientRect(Form1.hWnd, winrect)

' Loop through each pixel within Form1.
For y = 0 To winrect.bottom - winrect.top
    For x = 0 To winrect.right - winrect.left
        ' Select a random color by choosing a value between 0 and 255 ' inclusive for each component of the color.
        rgbval = RGB(Int(256 * Rnd), Int(256 * Rnd), Int(256 * Rnd))
        ' Set the pixel to the color above.
        retval = SetPixelV(Form1.hDC, x, y, rgbval)
    Next x
Next y

See Also

GetPixel, SetPixel

Category

Bitmaps

Go back to the alphabetical Function listing.
SetProp Function

Declare Function SetProp Lib "user32.dll" Alias "SetPropA" (ByVal hWnd As Long, ByVal lpString As String, ByVal hData As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetProp sets the value of one of the window properties of a window. If the window property does not already exist, this function will also create it. If the window property already exists, this function merely changes its value. Any window properties which your program creates must eventually be removed via the RemoveProp function.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to set a window property of.
**lpString**

The name of the window property to set.

**hData**

A handle to the data to set as the window property. This handle can refer to anything you wish.

## Example

```vbnet
' This code is licensed according to the terms and conditions listed here.

' Set the "LookupFile" property of window Form1 to a string.
' This example also shows how to remove the property.
Dim hStr As Long, pStr As Long ' handle and pointer to the string
Dim thevalue As String ' the string referenced by the handle
Dim retval As Long ' return value

' Set the value of the string (this could be anything, really).
thevalue = "C:\Icons\default.ico"
' Create a memory block...
hStr = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(thevalue))
' ...and copy the string into it.
pStr = GlobalLock(hStr) ' get a pointer to the block
retval = lstrcpy(pStr, text) ' copy the string
retval = GlobalUnlock(hStr) ' release the pointer

' The handle hStr now refers to a memory block holding the string. Set
' the "LookupFile" property to this memory block.
retval = SetProp(Form1.hWnd, "LookupFile", hStr)
' Note how we cannot yet free the memory block since it is still in use.

' *** INSERT OTHER CODE (such as the GetProp example) HERE ***

' The following code releases the "LookupFile" property and frees
' the memory block to which it points.
' (this code assumes the same Dims as above)
hStr = RemoveProp(Form1.hWnd, "LookupFile")
' The property is gone; now free the memory block.
retval = GlobalFree(hStr)
```

## See Also

GetProp, RemoveProp
SetPolyFillMode Function

Declare Function SetPolyFillMode Lib "gdi32.dll" (ByVal hdc As Long, ByVal nPolyFillMode As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetPolyFillMode sets the method used by a device to fill polygonal areas and shapes. The two modes differ only over complex overlapping polygons, where the edges of the polygon criss-cross each other inside the interior (see the example for an example of a complex overlapping polygon). The function returns 0 if an error occurred, or exactly one of the polygon fill mode flags specifying the filling mode the device had previously been set to.

hdc
A device context to the device to set the polygon filling mode of.

nPolyFillMode
Exactly one of the following flags specifying the polygon filling mode to use:

ALTERNATE = 1
The device alternates between filling and not filling contiguous sections whose boundaries are determined by the edge(s) of the polygon crossing through the polygon's interior.

WINDING = 2
Any section inside the polygon is filled, regardless of any intra-polygonal boundaries and edges.

Example:

' Use the alternating fill mode to fill that five-pointed star that everyone knows so well. Alternating filling tells window Form1 not to fill in the pentagonal interior ' of the star. It's hard to describe, so just run the example!
Dim points(0 to 4) As POINT_TYPE   ' points of the star
Dim retval As Long       ' return value

' Load the coordinates of a somewhat distorted five-pointed "pentagram" star ' design into the array.
points(0).x = 200: points(0).y = 400   ' 1st point: (200,400)
points(1).x = 300: points(1).y = 200   ' 2nd point: (300,200)
points(2).x = 400: points(2).y = 400   ' 3rd point: (400,400)
points(3).x = 150: points(3).y = 300   ' 4th point: (150,300)
points(4).x = 450: points(4).y = 300   ' 5th point: (450,300)
' Draw the polygon using alternating fill mode
retval = SetPolyFillMode(Form1.hDC, ALTERNATE)  ' set the fill mode
retval = Polygon(Form1.hDC, points(0), 5)  ' draw the polygon

See Also: GetPolyFillMode
Category: Regions

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetRect Function

Declare Function SetRect Lib "user32.dll" (lpRect As RECT, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetRect sets the position and size of a rectangle. The two coordinates specified are the upper-left and lower-right corners of the rectangle. The function returns 0 if an error occurred, or 1 if successful.

lpRect
  The rectangle to set the position and size of.
X1
  The x coordinate of the upper-left corner of the rectangle.
Y1
  The y coordinate of the upper-left corner of the rectangle.
X2
  The x coordinate of the lower-right corner of the rectangle.
Y2
  The y coordinate of the lower-right corner of the rectangle.

Example:

' Set rectangle r to represent the rectangle (20,30)-(100,50). Note
' that using this function is more efficient and takes less room than setting the
' rectangle's
' four member values individually.
Dim r As RECT  ' the rectangle to set
Dim retval As Long  ' return value
retval = SetRect(r, 20, 30, 100, 50)  ' r now represents (20,30)-(100,50)

See Also: SetRectEmpty
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetRectEmpty Function

Declare Function SetRectEmpty Lib "user32.dll" (lpRect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetRectEmpty sets a rectangle to an empty state. An empty rectangle is one that has a nonpositive width and/or height. In this case, Windows sets the rectangle to (0,0)-(0,0). The function returns 0 if an error occurred, or 1 if successful.

lpRect
The rectangle to set as empty.

Example:

' Set rectangle r to empty. Note this is much easier than manually
' setting its four member values to 0 individually.
Dim r As RECT  ' the rectangle to make empty
Dim retval As Long  ' return value

retval = SetRectEmpty(r)  ' r now equals (0,0)-(0,0)

See Also: IsRectEmpty, SetRect
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetSysColors Function

Declare Function SetSysColors Lib "user32.dll" (ByVal cElements As Long, lpaElements As Long, lpaRgbValues As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetSysColors changes the system colors used by Windows. Windows uses these colors when displaying the typical widgets such as title bars, scroll bars, the desktop, menus, etc. This function can set multiple different system colors simultaneously, by passing all the new RGB color values in an array. SetSysColors also notifies all windows of the change, so the color change takes effect immediately. However, the new colors are not saved after Windows shuts down.

Return Value

If successful, the function returns a nonzero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- cElements
  The number of elements in the arrays passed as lpaElements and lpaRgbValues.

- lpaElements
  An array holding the identifiers of all the system colors to change. Each element can be one of the following flags, specifying a system color:
  - COLOR_3DDKSHADOW
    The dark shadow color for 3D objects.
  - COLOR_3DFACE, COLOR_BTNFACE
    The face color for 3D objects.
COLOR_3DHILIGHT, COLOR_3DHIGHLIGHT, COLOR_BTNHILIGHT, COLOR_BTNHIGHLIGHT  
   The highlight (opposite of shadow) color for 3D objects.
COLOR_3DLIGHT  
   The light (opposite of shadow) color for 3D objects.
COLOR_3DSHADOW, COLOR_BTNSHADOW  
   The shadow color for 3D objects.
COLOR_ACTIVEBORDER  
   The active window border color.
COLOR_ACTIVECAPTION  
   The active window title bar color. Windows 98, 2000: The color of the left side of the active window title bar gradient, if the gradient effect is used.
COLOR_APPWORKSPACE  
   The background color of multiple document interface (MDI) windows.
COLOR_BACKGROUND, COLOR_DESKTOP  
   The desktop color.
COLOR_BTNTEXT  
   The text color for pushbuttons.
COLOR_CAPTIONTEXT  
   The color of window caption text, size boxes, and scroll bar arrow boxes.
COLOR_GRADIENTACTIVECAPTION  
   Windows 98, 2000: The color of the right side of the active window title bar gradient, if the gradient effect is used.
COLOR_GRADIENTINACTIVECAPTION  
   Windows 98, 2000: The color of the right side of an inactive window's title bar gradient, if the gradient effect is used.
COLOR_GRAYTEXT  
   The color for disabled (grayed-out) text.
COLOR_HIGHLIGHT  
   The color used to highlight selected items.
COLOR_HIGHLIGHTTEXT  
   The color used for the text of highlighted items.
COLOR_HOTLIGHT  
   Windows 98, 2000: The color of a hot-tracked item, which is executed with a single click.
COLOR_INACTIVEBORDER  
   The color of an inactive window's border.
COLOR_INACTIVECAPTION  
   The color of an inactive window's caption. Windows 98, 2000: The color of the right side of an inactive window's title bar gradient, if the gradient effect is used.
COLOR_INACTIVECAPTIONTEXT  
   The color of an inactive window's caption text.
COLOR_INFOBK  
   The background color for tooltop controls.
COLOR_INFOTEXT  
   The text color for tooltip controls.
COLOR_MENU  
   The background color of menus.
COLOR_MENUTEXT  
   The color of menu text.
COLOR_SCROLLBAR  
   The color of a scroll bar's gray area.
COLOR_WINDOW  
   The background color of a window.
COLOR_WINDOWFRAME
The color of a window frame.

COLOR_WINDOWTEXT
The color of text in a window.

*pRgbValues
An array of the RGB values of the new system colors to assign. These elements correspond directly to the elements in *pElements. For example, the first element of this array is the color to use for the system color identified by the first element of *pElements, etc.

Constant Definitions

Const COLOR_3DDKSHADOW = 21
Const COLOR_3DFACE = COLOR_BTNPFACE
Const COLOR_3DHIGHLIGHT = COLOR_BTNHIGHLIGHT
Const COLOR_3DLIGHT = 22
Const COLOR_3DSHADOW = COLOR_BTNSHADOW
Const COLOR_ACTIVEBORDER = 10
Const COLOR_ACTIVECAPTION = 2
Const COLOR_APPWORKSPACE = 12
Const COLOR_BACKGROUND = 1
Const COLOR_BTNPFACE = 15
Const COLOR_BTNHIGHLIGHT = 20
Const COLOR_BTNHILIGHT = COLOR_BTNHIGHLIGHT
Const COLOR_BTNSHADOW = 16
Const COLOR_BTNPTEXT = 18
Const COLOR_CAPTIONTEXT = 9
Const COLOR_DESKTOP = COLOR_BACKGROUND
Const COLOR_GRADIENTACTIVECAPTION = 27
Const COLOR_GRADIENTINACTIVECAPTION = 28
Const COLOR_GRAYTEXT = 17
Const COLOR_HIGHLIGHT = 13
Const COLOR_HIGHLIGHTTEXT = 14
Const COLOR_HOTLIGHT = 26
Const COLOR_INACTIVEBORDER = 11
Const COLOR_INACTIVECAPTION = 3
Const COLOR_INACTIVECAPTIONTEXT = 19
Const COLOR_INFOBK = 24
Const COLOR_INFOTEXT = 23
Const COLOR_MENU = 4
Const COLOR_MENUTEXT = 7
Const COLOR_SCROLLBAR = 0
Const COLOR_WINDOW = 5
Const COLOR_WINDOWFRAME = 6
Const COLOR_WINDOWTEXT = 8

Example

Reverse the gradient colors in windows' title bars. In other words, the left-side gradient color is swapped with the right-side gradient color, for both active and inactive windows. After making the change, a notification is sent to all windows, to insure
that the change takes effect immediately. Of course, this example won't work properly on Windows 95 or Windows NT 3.1 through 4.0, but you can still see how these two functions are used by looking at the source code.

This example runs when the user clicks command button Command1. So, to use this example, you must first place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed [here].

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function GetSysColor Lib "user32.dll" (ByVal nIndex As Long) As Long
Public Declare Function SetSysColors Lib "user32.dll" (ByVal cElements As Long, lpaElements As Long, _
    lpaRgbValues As Long) As Long
Public Const COLOR_ACTIVECAPTION = 2
Public Const COLOR_GRADIENTACTIVECAPTION = 27
Public Const COLOR_GRADIENTINACTIVECAPTION = 28
Public Const COLOR_INACTIVECAPTION = 3

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim activeLeftColor As Long     ' color of left-side gradient of active
    Dim activeRightColor As Long    ' color of right-side gradient of active
    Dim inactiveLeftColor As Long   ' color of the left-side gradient of inactive
    Dim inactiveRightColor As Long  ' color of the right-side gradient of inactive
    Dim colorNames(0 To 3) As Long  ' identifiers of the system colors to change
    Dim colorRGBs(0 To 3) As Long   ' RGB values of the system colors to change
    Dim retval As Long              ' generic return value

    ' Get the RGB values of the colors used in title bars.
    activeLeftColor = GetSysColor(COLOR_ACTIVECAPTION)
    activeRightColor = GetSysColor(COLOR_GRADIENTACTIVECAPTION)
    inactiveLeftColor = GetSysColor(COLOR_INACTIVECAPTION)
    inactiveRightColor = GetSysColor(COLOR_GRADIENTINACTIVECAPTION)

    ' Load the arrays with the new values to assign. Note how we're switching
    ' the values used on the left and right sides of the two gradients.
    colorNames(0) = COLOR_ACTIVECAPTION
    colorRGBs(0) = activeRightColor
    colorNames(1) = COLOR_GRADIENTACTIVECAPTION
    colorRGBs(1) = activeLeftColor
    colorNames(2) = COLOR_INACTIVECAPTION
    colorRGBs(2) = inactiveRightColor
    colorNames(3) = COLOR_GRADIENTINACTIVECAPTION

    SetSysColors 5, Array(activeLeftColor, activeRightColor, inactiveLeftColor, inactiveRightColor), Array(colorNames, colorRGBs)
End Sub
colorRGs(3) = inactiveLeftColor

' Change the system color settings as specified above.
retval = SetSysColors(4, colorNames(0), colorRGs(0))

End Sub

See Also

GetSysColor

Category

System Information

Back to the Function list.
Back to the Reference section.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setsyscolors.html
SetSystemCursor Function

Declare Function SetSystemCursor Lib "user32.dll" (ByVal hcur As Long, ByVal id As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetSystemCursor changes one of the cursors that Windows provides. For example, this function can change the cursor used to represent the default arrow cursor. Be careful using this function, since this redefines the default cursors instead of simply setting the current look of the cursor. The function destroys the cursor handle passed to it once it sets the new default cursor. The function returns 1 if successful, or 0 if an error occured.

hcur
A handle to the cursor to use as the new default cursor for a given type of cursor. The function destroys this handle once the new cursor is set.

id
Exactly one of the following flags specifying which of the Windows default cursors to redefine:
OCR_APPSTARTING = 32650
The application starting (arrow and hourglass) cursor.
OCR_CROSS = 32515
The cross-shaped cursor.
OCR_IBEAM = 32513
The text selection (I-beam) cursor.
OCR_ICON = 32641
Win NT only: The empty icon cursor.
OCR_NO = 32648
The "no"-symbol (circle with slash through it) cursor.
OCR_NORMAL = 32512
The normal arrow cursor.
OCR_SIZE = 32640
Win NT only: The four-arrow resize/move cursor.
OCR_SIZEALL = 32646
The four-arrow resize/move cursor.
OCR_SIZENESW = 32643
The double-arrow resize/move cursor pointing to the upper-right and lower-left.
OCR_SIZENS = 32645

The double-arrow resize/move cursor pointing up and down.
OCR_SIZENWSE = 32642
   The double-arrow resize/move cursor pointing to the upper-left and lower-right.
OCR_SIZEWE = 32644
   The double-arrow resize/move cursor pointing left and right.
OCR_UP = 32516
   The up arrow cursor.
OCR_WAIT = 32514
   The waiting (hourglass) cursor.

Example:

' Set Windows's default "hourglass" cursor to the cursor in
' C:\MyProg\NewWait.ani.
Dim hcursor As Long  ' receives handle to the cursor from the file
Dim retval As Long  ' return value

' Load the desired cursor from the file:
hcursor = LoadCursorFromFile("C:\MyProg\NewWait.ani")
retval = SetSystemCursor(hcursor, OCR_WAIT)  ' redefine hourglass cursor
' Now Windows will use NewWait.ani as the hourglass cursor.

See Also: SetCursor
Category: Cursor

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetSystemTime Function

Declare Function SetSystemTime Lib "kernel32.dll" (lpSystemTime As SYSTEMTIME) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetSystemTime sets the date and time of the system's clock. The time given to the function must be in UTC time (Coordinated Universal Time, a.k.a. Greenwich Mean Time (GMT)), not in the system's local time zone.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpSystemTime

The new time and date to use to set the system clock. The wDayOfWeek member of the structure is ignored.
Example

' This code is licensed according to the terms and conditions listed here.

' Set the system clock to the following time:
' October 3, 1999 16:46:00 UTC
Dim systime As SYSTEMTIME ' new time and date
Dim retval As Long ' return value

' Put the date information into the structure.
systime.wYear = 1999
systime.wMonth = 10
systime.wDay = 3
systime.wHour = 16
systime.wMinute = 46
systime.wSecond = 0
systime.wMilliseconds = 0

' Set the system clock to that time and date.
retval = SetSystemTime(systime)

See Also

GetSystemTime

Category

Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 3, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setsystemtime.html
SetTextAlign Function

Declare Function SetTextAlign Lib "gdi32.dll" (ByVal hdc As Long, ByVal wFlags As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetTextAlign specifies how a device displays text relative to a given reference point. The reference point is the point used to identify where a line of text should be written. The function returns 1 if successful, or 0 if an error occurred.

hdc
The device context of the device to set the reference point alignment of.

wFlags
Up to three of the following flags specifying where the text will be written in relation to a given reference point. Only one flag may be specified for horizontal alignment, vertical alignment, and current point updating.

- TA_BASELINE = 24
  The reference point will be on the baseline of the text.
- TA_BOTTOM = 8
  The reference point will be on the bottom edge of the bounding rectangle of the text.
- TA_CENTER = 6
  The reference point will be horizontally centered along the bounding rectangle of the text.
- TA_LEFT = 0
  The reference point will be on the left edge of the bounding rectangle of the text.
- TA_NOUPDATECP = 0
  Do not set the current point to the reference point.
- TA_RIGHT = 2
  The reference point will be on the right edge of the bounding rectangle of the text.
- TA_RTLREADING = 256
  Win 95/98 only: Display the text right-to-left (if the font is designed for right-to-left reading).
- TA_TOP = 0
  The reference point will be on the top edge of the bounding rectangle of the text.
- TA_UPDATECP = 1
  Set the current point to the reference point.

Example:

' Display the text "Hello, world!" on window Form1 at (100,50).
' Center the text horizontally at that point and have it appear below the point.
Dim retval As Long ' return value

' Set the reference point to be centered horizontally and on the top edge of the text:
retval = **SetTextAlign**(Form1.hDC, TA_CENTER Or TA_TOP Or TA_NOUPDATECP)
' Display the text:
retval = **TextOut**(Form1.hDC, 100, 50, "Hello, world!", 13)

**See Also:** [GetTextAlign](#), **TextOut**

**Category:** Fonts & Text

---

This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](#).

Go back to the Windows API Guide home page.

E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at [http://www.vbapi.com/ref/s/settextalign.html](http://www.vbapi.com/ref/s/settextalign.html)
SetThreadLocale Function

Declare Function SetThreadLocale Lib "kernel32.dll" (ByVal Locale As Long) As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Not Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetThreadLocale chooses which locale to assign to the thread calling the function (i.e., the thread from your program). This locale should then be used whenever the program displays certain data to the user.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

Locale

The identifier of the locale to assign to the calling thread. This can also be one of the following flags:

- LOCALE_SYSTEM_DEFAULT
  - The system's default locale.
- LOCALE_USER_DEFAULT
  - The user's default locale.
Constant Definitions

Const LOCALE_SYSTEM_DEFAULT = &H400
Const LOCALE_USER_DEFAULT = &H800

Example

' This code is licensed according to the terms and conditions listed here.
' Assign the system's current locale to this thread. Note that
' this could be significantly different than the locale previously
' assigned to the thread! Then use the newly selected locale
' to display the current date.
Dim retval As Long  ' return value
Dim today As SYSTEMTIME  ' today's date and time
Dim datestr As String  ' receives the formatted date string
Dim strlen As Long  ' length of the buffer for the formatted date string

' First of all, set the locale to the system's default.
retval = SetThreadLocale(LOCALE_SYSTEM_DEFAULT)

' Get today's date and time in the local time zone.
GetLocalTime today

' Make sufficient room in the buffer to receive the date string.
datestr = Space(255)
' Format today's date as a Long Date.
strlen = GetDateFormat(0, DATE_LONGDATE, today, CLng(0), datestr, Len(datestr))
' Remove the empty space from the formatted date string.
datestr = Left(datestr, strlen)
' Display today's date as a Long Date.
Debug.Print "Today is "; datestr

See Also

GetThreadLocale

Category

National Language Support

Go back to the alphabetical Function listing.
Go back to the Reference section index.


**SetTimer Function**

Declare Function SetTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc As Long) As Long

**Platforms**

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

**Description & Usage**

The `SetTimer` function creates a timer that triggers an event after so many milliseconds have elapsed. The timer continues triggering events between intervals until it is removed using `KillTimer`. The timer can be configured to either send a `WM_TIMER` message to the owning window or call a **callback function** whenever the time-out period elapsed.

**Return Value**

If successful, the function returns a unique integer identifying the created timer. If an error occurred, the function returns 0 (call `GetLastError` to get the error code).

**Visual Basic-Specific Issues**

None.

**Parameters**

- **hWnd**
  - A handle to the window that will own the timer. To make the timer unassociated with any window, set this to 0.
- **nIDEvent**
  - A nonzero value that will be used to identify the timer. If `hWnd` is 0, this parameter is ignored.
- **uElapse**
  - The time-out period, in milliseconds, to elapse between timer notifications.
- **lpTimerFunc**
  - A pointer to the **TimerProc** callback function to call whenever the time-out period elapses. If this is 0, a `WM_TIMER` message is instead sent to the window specified by `hWnd` when the time-out period elapses.
Example

Display the current time in text box control Text1. The time is updated twice every second, and the time is formatted according to the current locale's settings. To use this example, place a text edit box named Text1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type
Public Declare Function SetTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc As Long) As Long
Public Declare Function KillTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long) As Long
Public Declare Function GetTimeFormat Lib "kernel32.dll" Alias "GetTimeFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpTime As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpTimeStr As String, ByVal cchTime As Long) As Long

' *** Place the following code inside a module. ***

' The following function will execute twice every second. It retrieves the current time and displays it according to the current locale's formatting preferences.
Public Sub TimerProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal idEvent As Long, ByVal dwTime As Long)
    Dim systime As SYSTEMTIME    ' the current time
    Dim timestr As String * 260    ' receives the formatted string
    Dim slength As Long    ' length of formatted string returned

    ' Retrieve the current time, according to the computer's time zone.
    GetLocalTime systime
    ' Format a string to represent the time.
    slength = GetTimeFormat(0, 0, systime, CLng(0), timestr, Len(timestr))
    ' Display the string in Text1, found on window Form1.
    Form1.Text1.Text = Left(timestr, slength)
End Function

' *** Place the following code inside Form1. ***
Create the timer when the form opens and destroy it when the form closes. The timer is given an ID of 1, so the return values don't need to be saved.

Private Sub Form_Load()
    Dim retval As Long ' return value
    retval = SetTimer(Form1.hWnd, 1, 500, AddressOf TimerProc)
End Sub

Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = KillTimer(Form1.hWnd, 1)
End Sub

See Also

KillTimer

Category

Timers

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/settimer.html
SetVolumeLabel Function

Declare Function SetVolumeLabel Lib "kernel32.dll" Alias "SetVolumeLabelA" (ByVal lpRootPathName As String, ByVal lpVolumeName As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

SetVolumeLabel sets the label of a file system volume. The new volume label must comply with length constaints and cannot contain any invalid characters.

Return Value

If an error occured, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

lpRootPathName

The path of the root directory of the drive to set the file system volume label of. This path must include the trailing backslash. For example, the C: drive would be "C:\".

lpVolumeName

The new name to assign to the volume.

Example

' This code is licensed according to the terms and conditions listed here.
Declarations and such needed for the example:
' (Copy these to the (declarations) section of a module.)
Public Declare Function SetVolumeLabel Lib "kernel32.dll" Alias "SetVolumeLabelA" _
    (ByVal lpRootPathName As String, ByVal lpVolumeName As String) As Long

' Set the file system volume label of the C: drive to "MYDRIVE".
Dim retval As Long  ' return value

' Set the new volume label.
retval = SetVolumeLabel("C:", "MYDRIVE")

See Also

GetVolumeInformation

Category

File System

Back to the Function list,
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/setvolumelabel.html
SetWindowLong Function

Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetWindowLong sets a 32-bit value constituting the information about a window. This function can also set a 32-bit value within the block of extra memory given to the window, if such a block exists.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns the previous setting of whatever 32-bit value was replaced.

Visual Basic-Specific Issues

None.

Parameters

hWnd

A handle to the window to set a 32-bit value in.

nIndex

To set a 32-bit value within the window's extra memory block, this is the zero-based offset of the byte to begin writing to. Valid values range from 0 to the size of the extra memory block in bytes minus four. To set a 32-bit value of one of the properties of the window, this is one of the following flags specifying which piece of information to set:

- GWL_EXSTYLE
  Set the extended window styles of the window. dwNewLong is the new setting.

- GWL_HINSTANCE
  Set which application instance is considered to own the window. dwNewLong is the new setting.
GWL_HWNDPARENT
Retrieve a handle to the parent window, if any. \textit{dwNewLong} is a handle to the instance to set as the owner.

GWL_ID
Set the identifier of the window. \textit{dwNewLong} is the new identifier.

GWL_USERDATA
Set the application-defined 32-bit value associated with the window. \textit{dwNewLong} is the new value.

GWL_STYLE
Retrieve the \textit{window styles} of the window.

GWL_WNDPROC
Set the \textit{WindowProc} hook function to use as the window's procedure. \textit{dwNewLong} is a pointer to the hook function to set as the window procedure.

If the window happens to be a dialog box, this could also be one of the following flags:

DWL_DLGPROC
Set the \textit{WindowProc} hook function to use as the dialog box procedure. \textit{dwNewLong} is a pointer to the hook function to set as the window procedure.

DWL_MSGRESULT
Set the return value of the last \textit{message} processed by the dialog box. \textit{dwNewLong} is the new value.

DWL_USER
Set the application-defined 32-bit value associated with the dialog box. \textit{dwNewLong} is the new value.

\textit{dwNewLong}
The 32-bit value to set as specified by \textit{nIndex}.

**Constant Definitions**

Const GWL_EXSTYLE = -20
Const GWL_HINSTANCE = -6
Const GWL_HWNDPARENT = -8
Const GWL_ID = -12
Const GWL_STYLE = -16
Const GWL_USERDATA = -21
Const GWL_WNDPROC = -4
Const DWL_DLGPROC = 4
Const DWL_MSGRESULT = 0
Const DWL_USER = 8

**Example**

Have a window play the SystemAsterisk sound whenever it gains or loses the focus. This is done by playing the sound whenever the window receives the WM_ACTIVATE message. To use this example, copy the following code into the proper locations.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, _
                         ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = -4
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Declare Function PlaySound Lib "winmm.dll" Alias "PlaySoundA" (ByVal lpszName As Any, ByVal hModule As Long, ByVal dwFlags As Long) As Long

Public Const WM_ACTIVATE = &H6

' *** Place this code in a module. ***

' The following variable is accessible to all code in this example.
Public pOldProc As Long ' pointer to the previous window function

' Define the new window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim retval As Long ' return value
    ' If the message is WM_ACTIVATE (we don't care about the parameters),
    ' play the SystemAsterisk sound.
    If uMsg = WM_ACTIVATE Then
        retval = PlaySound("SystemAsterisk", 0, SND_ALIAS Or SND_ASYNC)
    End If
    ' No matter what happened, use the old window procedure to
    ' finish processing the message.
    retval = CallWindowProc(pOldProc, hWnd, uMsg, wParam, lParam)
    ' Have this function return whatever the function above returned.
    WindowProc = retval
End Function

' *** Place the following code in the form window. ***

Private Sub Form_Load ()
    Dim retval As Long ' return value
    ' Set the new window procedure for Form1, saving a pointer to the old one.
    pOldProc = SetWindowLong(Me.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

Private Sub Form_Unload (Cancel As Integer)
    Dim retval As Long ' return value
    ' Restore the window's procedure before closing.
    retval = SetWindowLong(Me.hWnd, GWL_WNDPROC, pOldProc)
End Sub

See Also

GetObjectLong, SetClassLong
SetWindowPos Function

Declare Function SetWindowPos Lib "user32.dll" (ByVal hwnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SetWindowPos moves a window to a new location on the screen. Its physical coordinates, dimensions, and Z-order position (the Z-order determines which windows are on top of others) can be set. The function returns 0 if an error occurred or 1 if successful.

hwnd
  The handle of the window to move.

hWndInsertAfter
  Either the handle of the window to position this window behind, or exactly one of the following flags stating where in the Z-order to put the window:
  
  HWND_BOTTOM = 1
  
  Put the window at the bottom of the Z-order.

  HWND_NOTOPMOST = -2
  
  Put the window below all topmost windows and above all non-topmost windows.

  HWND_TOP = 0
  
  Put the window at the top of the Z-order.

  HWND_TOPMOST = -1
  
  Make the window topmost (above all other windows) permanently.

x
  The x coordinate of where to put the upper-left corner of the window.

y
  The y coordinate of where to put the upper-left corner of the window.

cx
  The x coordinate of where to put the lower-right corner of the window.

cy
  The y coordinate of where to put the lower-right corner of the window.

wFlags
  Zero or more of the following flags stating how to move the window:

  SWP_DRAWFRAME = &H20
  
  Same as SWP_FRAMECHANGED.

  SWP_FRAMECHANGED = &H20
  
  Fully redraw the window in its new position.

  SWP_HIDEWINDOW = &H80
  
  Hide the window from the screen.

  SWP_NOACTIVATE = &H10
  
  Do not make the window active after moving it unless it was already the active window.

  SWP_NOCOPYBITS = &H100
  
  Do not redraw anything drawn on the window after it is moved.
SWP_NOMOVE = &H2
Do not move the window.
SWP_NOSIZE = &H1
Do not resize the window.
SWP_NOREDRAW = &H8
Do not remove the image of the window in its former position, effectively leaving a ghost image on the screen.
SWP_NOZORDER = &H4
Do not change the window's position in the Z-order.
SWP_SHOWWINDOW = &H40
Show the window if it is hidden.

Example:

' Move window Form1 to the upper-left corner of the screen and make
' it appear above all other windows permanently. Note how the function is told not
' to resize the window, so we don't have to worry about the lower-right coordinate.
Dim flags As Long  ' the flags specifying how to move the window
Dim retval As Long  ' return value

' Do not resize the window, redraw the window in its new location
flags = SWP_NOSIZE Or SWP_DRAWFRAME
retval = SetWindowPos(Form1.hWnd, HWND_TOPMOST, 0, 0, 1, 1, flags)  ' move the window

See Also: BringWindowToTop, GetWindowRect, MoveWindow
Category: Windows

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SetWindowRgn Function

Declare Function SetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long, ByVal bRedraw As Boolean) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.51 or later.
- Windows CE: Not Supported.

Description & Usage

SetWindowRgn changes the visible region of a window. Using this function, you can make a window appear non-rectangular. Any portion of the window lying outside of the region is not drawn, and so is invisible.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (Windows NT/2000: use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

hWnd
A handle to the window to set the region of.

hRgn
A handle to a region that defines what to make the visible area of the window. If this is zero, the entire window is displayed (effectively removing any special region set previously).

bRedraw
Specifies whether to immediately redraw the window using its new region or not. If this is True, the window is redrawn. If this is False, it is not.

Example
Windows API Guide: SetWindowRgn Function

Use an elliptic region to make window Form1 appear elliptical. Notice how, as this example is written, a portion of the title bar is still visible after applying the region. This allows us to move the window without adding any special code providing a different way for the user to move the window. Normally, in a real program, you would not want the title bar to be displayed, but after all, this is just an example. At least it shows you that the region only changes what part of the window you can see -- it doesn’t change anything else about the window.

To run this example, make two windows Form1 and Form2. The latter will be used to illustrate GetWindowRgn. Place the following three buttons on Form1: a button labeled "Apply Region" and named cmdApplyRegion, a button labeled "Remove Region" and named cmdRemoveRegion, and a button labeled "Show Region" and named cmdShowRegion. Place these buttons near the center of Form1 to make sure parts of them won’t be hidden when the region is applied.

---

This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Declare Function GetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long) As Long
Public Declare Function SetWindowRgn Lib "user32.dll" (ByVal hWnd As Long, ByVal hRgn As Long, ByVal bRedraw As Boolean) As Long
Public Declare Function CreateEllipticRgn Lib "gdi32.dll" (ByVal nLeftRect As Long, ByVal nTopRect As Long, ByVal nRightRect As Long, ByVal nBottomRect As Long) As Long
Public Declare Function DeleteObject Lib "gdi32.dll" (ByVal hObject As Long) As Long
Public Declare Function InvertRgn Lib "gdi32.dll" (ByVal hdc As Long, ByVal hrgn As Long) As Long

*** Place the following code inside Form1. ***

Stores a handle to the special window region, if it exists.
Private hRgnWindow As Long

Private Sub Form_Load()
    ' Open Form2 when this loads. Form2 is used to display a sort of "shadow" of the region used by Form1, although it really isn't a shadow effect.
    Form2.Show
    ' Initially disable the "Remove Region" button, since one hasn't yet been applied.
    cmdRemoveRegion.Enabled = False
End Sub

Private Sub cmdApplyRegion_Click()
    ' Create an elliptic region slightly smaller than the current size of Form1, and make that the window region. This makes the previously rectangular Form1 appear to be an ellipse.
    Dim retval As Long  ' return value
    ' First, make the elliptical region, slightly smaller than Form1.
    hRgnWindow = CreateEllipticRgn(5, 5, (Form1.Width / Screen.TwipsPerPixelX) -

(Form1.Height / Screen.TwipsPerPixelY) - 5)
' Apply this region to Form1 and show the change immediately.
retval = SetWindowRgn(Form1.hWnd, hRgnWindow, True)

' To make sure that multiple regions aren't created, disable
' this button and enable the "Remove Region" button.
cmdApplyRegion.Enabled = False
cmdRemoveRegion.Enabled = True
End Sub

Private Sub cmdRemoveRegion_Click()
' Remove the window region from Form1, returning it to its
' normal rectangular shape.
Dim retval As Long  ' return value

' Set a null window region, which removes the current one entirely.
retval = SetWindowRgn(Form1.hWnd, 0, True)
' Delete the region object because it is no longer needed.
retval = DeleteObject(hRgnWindow)

' Since the region no longer exists, enable "Apply Region" and
' disable this button.
cmdApplyRegion.Enabled = True
cmdRemoveRegion.Enabled = False
End Sub

Private Sub cmdShowRegion_Click()
' Show the region currently applied to Form1 by inverting that region on
' Form2. Note that if no region is applied, nothing appears to happen
' because
' the actual window region is empty.
Dim hRgnCopy As Long  ' region that receives copy of Form1's region
Dim retval As Long    ' return value

' Create a region. It doesn't matter what, since it will be overwritten
' when GetWindowRgn is called. We just need to have some region.
hRgnCopy = CreateEllipticRgn(0, 0, 0, 0)
' Copy Form1's region to hRgnCopy. The actual value of the handle
' does not change, but the information it "points" to does.
retval = GetWindowRgn(Form1.hWnd, hRgnCopy)
' Invert the colors on Form2 that lie within this region.
retval = InvertRgn(Form2.hDC, hRgnCopy)
' Delete the copied region, since we no longer need it.
retval = DeleteObject(hRgnCopy)
End Sub

See Also

GetWindowRgn

SetWindowText Function

Declare Function SetWindowText Lib "user32.dll" Alias "SetWindowTextA"
(ByVal hWnd As Long, ByVal lpString As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SetWindowText changes the text of a given window. Although this function can set the text of regular windows owned by other programs, it cannot change the text of a control owned by a different program. To set the text of those controls, use the WM_SETTEXT message instead.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Parameters

hWnd
A handle to the window to set the text of.

lpString
The string to set as the window's text.

Example
' Set the title bar of window Form1 to "Generic Application 1.0".
Dim retval As Long  ' return value

retval = SetWindowText(Form1.hWnd, "Generic Application 1.0")

See Also

GetWindowText, WM_SETTEXT

Category

Windows
SHAddToRecentDocs Function

Declare Sub SHAddToRecentDocs Lib "shell32.dll" (ByVal uFlags As Long, ByVal pv As Any)

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**SHAddToRecentDocs** adds a shortcut to a file to the shell's Documents menu under the Start menu. To accomplish this, the function creates a shortcut to the file under the Recent directory (the folder identified by the CSIDL CSIDL_RECENT). Instead of adding a file to the Documents menu, this function can also clear the entire contents of the Documents menu.

Return Value

**SHAddToRecentDocs** does not return a value.

Visual Basic-Specific Issues

When passing 0 for the **pv** parameter, you must use the expression **CLng(0)** to pass it correctly.

Parameters

**uFlags**

Either one of the following flags specifying the nature of the **pv** parameter, or 0 to clear the contents of the Documents menu.

- **SHARD_PIDL**
pv is a **pointer** to an **ITEMIDLIST** structure (a PIDL) identifying the file to add to the Documents menu.

**SHARD_PATH**

pv is the filename of the file to add to the Documents menu.

pvIdentifies the file to add to the Documents menu. It is either a PIDL or a string, depending on the value passed as *uFlags*. If *uFlags* is 0, this must also be 0.

## Constant Definitions

```plaintext
Const SHARD_PIDL = 1
Const SHARD_PATH = 2
```

## Example

```plaintext
' This code is licensed according to the terms and conditions listed [here](http://www.vbapi.com/ref/s/shaddtorecentdocs.html).

' First, clear the Documents menu. Then add a link to the file C:\MyDocs\report.txt to the Documents menu. Of course, you should always ask the user before you clear the Documents menu!

' Clear the Documents menu entirely.
SHAddToRecentDocs 0, CLng(0)

' Then add the file C:\MyDocs\report.txt to the menu.
SHAddToRecentDocs SHARD_PATH, "C:\MyDocs\report.txt"
```

## Category

**Shell**

[Go back to the alphabetical Function listing.](http://www.vbapi.com/ref/s/index.html#s)

[Go back to the Reference section index.](http://www.vbapi.com/ref/index.html)
SHBrowseForFolder Function

Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpbi As BROWSEINFO) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

SHBrowseForFolder opens the Browse for Folder dialog box, asking the user to select a folder on the system. The folder selected can be either a physical path on a disk or a virtual folder. The PIDL obtained by the function, if any, must be freed by using the CoTaskMemFree function.

Return Value

If an error occurred or the user pressed Cancel, the function returns 0. If the user successfully selected a folder, the function returns a pointer to an ITEMIDLIST structure (a PIDL) identifying the selection.

Visual Basic-Specific Issues

None.

Parameters

lpbi

Holds all the settings necessary to initialize the Browse for Folder dialog box. This structure also receives some information from the function if successful.

Example

' This code is licensed according to the terms and conditions listed here.
Open the Browse for Folder dialog box and display both the display name and
the actual name of the folder (if it is not a virtual folder). Although any
folders under My Computer can be selected, have the directory
C:\StartHere selected by default.

*** Place the following code in a module. ***

This function compensates for the fact that the AddressOf operator
can only be used in a function call. It returns the parameter
passed to it.
Public Function DummyFunc(ByVal param As Long) As Long
    DummyFunc = param
End Function

This function is the callback function for the dialog box. It sets
the selected folder to C:\StartHere when the box is initialized.
Public Function BrowseCallbackProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal lParam As Long, ByVal lpData As Long) As Long
    Dim pathstring As String  ' name of path to set by default
    Dim retval As Long  ' return value

    ' If the BFFM_INITIALIZED message is received, set the
default selection to C:\StartHere.
    Select Case uMsg
        Case BFFM_INITIALIZED
            pathstring = "C:\StartHere"  ' the path to make the default selection
            ' Send a message to the dialog box telling it to select this path.
            ' Note the use of ByVal and the CLng function here.
            retval = SendMessage(hwnd, BFFM_SETSELECTION, ByVal CLng(1), ByVal pathstring)
    End Select
    BrowseCallbackProc = 0  ' the function should always return 0
End Function

*** Place the following code where you want to open the ***

*** Browse for Folder dialog box. ***
Dim bi As BROWSEINFO  ' structure passed to the function
Dim pidl As Long  ' PIDL to the user's selection
Dim physpath As String  ' string used to temporarily hold the physical path
Dim retval As Long  ' return value

' Initialize the structure to be passed to the function.
bi.hwndOwner = Form1.hWnd  ' window Form1 is the owner of the dialog box
' Specify the My Computer virtual folder as the root
retval = SHGetSpecialFolderLocation(Form1.hWnd, CSIDL_DRIVES, bi.pidlRoot)
' Make room in the buffer to get the [virtual] folder's display name
bi.pszDisplayName = Space(260)
bi.lpszTitle = "Please choose a folder."  ' Message displayed to the user
bi.ulFlags = 0  ' no flags are needed here
' Identify the callback function to use for the dialog box. Note
' how our DummyFunc is needed because AddressOf only works
' inside a function call.
browseCallbackProc = DummyFunc(AddressOf BrowseCallbackProc)
Windows API Guide: SHBrowseForFolder Function

bi.lParam = 0  ' the callback function here doesn't need this
bi.lImage = 0  ' this will be set by the function

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
  ' Remove the empty space from the display name variable.
  bi.pszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
  Debug.Print "The user selected: "; bi.pszDisplayName
  ' If the folder is not a virtual folder, display its physical location.
  physpath = Space(260)  ' make room in the buffer
  retval = SHGetPathFromIDList(pidl, physpath)
  If retval = 0 Then
    Debug.Print "Physical Location: (virtual folder)"
  Else
    ' Remove the empty space and display the result.
    physpath = Left(physpath, InStr(physpath, vbNullChar) - 1)
    Debug.Print "Physical Location: "; physpath
  End If
  ' Free the pidl returned by the function.
  CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to
' identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

Category

Shell

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 19, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shbrowseforfolder.html
Shell_NotifyIcon Function

Declare Function Shell_NotifyIcon Lib "shell32.dll" Alias "Shell_NotifyIconA" (ByVal dwMessage As Long, pnid As NOTIFYICONDATA) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Shell_NotifyIcon manipulates an icon located in the taskbar's status area, commonly referred to as the "system tray." This function adds a new icon, removes an existing icon, or changes an existing icon owned by the calling program. Programs typically use tray icons when they run in the background without a window; the icon gives the user a way to access the running program.

All of a tray icon's messages are sent to its owner window for processing. See the page for the NOTIFYICONDATA structure for more details on how this is done. The standard behavior of Shell_NotifyIcon used the following messages: WM_MOUSEMOVE, WM_LBUTTONDOWN, WM_LBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP, etc.

Return Value

The function returns 0 if an error occurred, or a non-zero value if successful.

Visual Basic-Specific Issues

None.

Parameters

**dwMessage**

One of the following flags specifying the action to take:

- **NIM_ADD**
  - Add the icon described by pnid to the system tray.
- **NIM_DELETE**
Remove the icon described by \textit{pnid} from the system tray.

\textbf{NIM\_MODIFY}
Modify the icon described by \textit{pnid}.

\textbf{NIM\_SETFOCUS}
\textbf{Windows 2000:} Return the focus to the system tray. This should be done after completing the user interface operations associated with the icon.

\textbf{NIM\_SETVERSION}
\textbf{Windows 2000:} Use the \textit{uVersion} member of the structure passed as \textit{pnid} to determine which taskbar behavior to use. The two options are the Windows 2000-specific behavior and the behavior found on other Windows platforms. Note that this guide does not explain how to use the behavior introduced by Windows 2000.

\textit{pnid}
Provides identifying information and settings for the tray icon, as needed by the value passed as \textit{dwMessage}.

\section*{Constant Definitions}

\begin{verbatim}
Const NIM_ADD = &H0
Const NIM_DELETE = &H2
Const NIM_MODIFY = &H1
Const NIM_SETFOCUS = &H4
Const NIM_SETVERSION = &H8
\end{verbatim}

\section*{Example}

\textbf{Note:} To use this example, you must first use the Menu Editor utility to create a small menu on Form1. Create a menu called \texttt{mnuTrayIconPopup} with its Visible check box cleared. Then, create two menu items under it: \texttt{mnuAbout} and \texttt{mnuExit}. You can give those items whatever captions you which; those are what will be displayed in the menu. If you do this properly, you will not see a menu bar in Form1 but the example code will function properly.

\begin{verbatim}
' This code is licensed according to the terms and conditions listed here.

' Place an icon in the system tray when window Form1 opens. When right-clicked, a small pop-up menu appears. The icon is deleted when Form1 closes.
' Notice how the window procedure for Form1 is extended in order to process the messages for the icon. Also, note the application-defined message used for tray icons.

' *** Place the following code in a module. ***
' Public variable definitions.
Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public Const PK\_TRAYICON = &H401  ' program-defined message for tray icon action

' This function acts as the new window procedure for Form1. It handles the program-defined PK\_TRAYICON message, used whenever a user event occurs with the tray icon.
Public Function WindowProc(ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Select Case uMsg
        Case PK\_TRAYICON
            ' If the user released the right mouse button over the icon, display the mnuTrayIconPopup menu at the mouse cursor's position. Make
\end{verbatim}

mnuAbout the default selection in the menu.
If lParam = WM_RBUTTONUP Then
    Form1.PopupMenu mnuTrayIconPopup, , , mnuAbout
End If
WindowProc = 1 ' this return value doesn't really matter
Case Else
' Pass the message to the procedure Visual Basic provided.
    WindowProc = CallWindowProc(pOldProc, hWnd, Msg, wParam, lParam)
End Select
End Function

' *** Place the following code in Form1. ***
' This function creates the tray icon and hooks up the window procedure
' when Form1 first opens.
Private Sub Form1_Open()
    Dim nid As NOTIFYICONDATA ' icon information
    Dim retval As Long ' return value

    ' Put the icon settings into the structure.
    With nid
       .cbSize = Len(nid) ' size of structure
       .hWnd = Form1.hWnd ' owner of the icon and processor of its messages
       .uID = 1 ' unique identifier for the window's tray icons
       .uFlags = NIF_ICON Or NIF_MESSAGE Or NIF_TIP ' provide icon, message, and tool
tip text
       .uCallbackMessage = PK_TRAYICON ' message to use for icon events
       .hIcon = Form1.Icon ' handle to the icon to actually display in the tray
       .szTip = "Sample Tray Icon" & vbNullChar ' tool tip text for icon
    End With

    ' Add the icon to the system tray.
    retval = Shell_NotifyIcon(NIM_ADD, nid)
    ' Set the new window procedure for window Form1.
    pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' This function removes the tray icon and returns the old window procedure to use.
Private Sub Form1_Unload(Cancel As Integer)
    Dim nid As NOTIFYICONDATA ' icon information
    Dim retval As Long ' return value

    ' Load the structure with just the identifying information.
    With nid
       .cbSize = Len(nid) ' size of structure
       .hWnd = Form1.hWnd ' handle of owning window
       .uID = 1 ' unique identifier
    End With

    ' Remove the icon from the system tray.
    retval = Shell_NotifyIcon(NIM_DELETE, nid)

    ' Make the old window procedure the current window procedure.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
End Sub

Category

Shell

Back to the Function list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shell_notifyicon.html
ShellExecute Function

Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal lpDirectory As String, ByVal nShowCmd As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

ShellExecute opens, prints, or executes a file using the Windows shell. When working with a non-executable file, the file is opened using its associated program. ShellExecute can also open Windows Explorer windows. The function returns immediately after opening the file, starting the program, or performing whatever other action was specified.

Return Value

If successful, the function returns a handle to the instance of the application which the function started. If an error occurred, the function returns either 0 (indicating a lack of memory) or one of the following flags:

- ERROR_BAD_FORMAT
  The specified executable file was somehow invalid.
- SE_ERR_ACCESSDENIED
  Access was denied.
- SE_ERR_ASSOCINCOMPLETE
  The filename association is either incomplete or invalid.
- SE_ERR_DDEBUSY
  The DDE action could not run because other DDE actions are in process.
- SE_ERR_DDEFAIL
  The DDE transaction failed.
- SE_ERR_DDETIMEOUT
  The DDE transaction was not completed because the request timed out.
- SE_ERR_DLLNOTFOUND
  A required DLL file could not be found.
- SE_ERR_FNF
  The file could not be found.
SE_ERR_NOASSOC
    There is no program associated with the specified type of file.
SE_ERR_OOM
    Windows has insufficient memory to perform the operation.
SE_ERR_PNF
    The path could not be found.
SE_ERR_SHARE
    A sharing violation occurred.

Visual Basic-Specific Issues

None.

Parameters

hwnd
    A handle to the window calling the function.
lpOperation
    The operation to perform on lpFile. This can be one of the following strings, although other options are possible, depending on the file being acted upon:
    "explore"
        If lpFile is a path name, open it in a Windows Explorer window.
    "open"
        Open lpFile using its associated program. Opening an executable file runs it. This is the default action if none is specified.
    "print"
        Print lpFile using its associated program.
lpFile
    The name of the file to open, print, execute, or whatever is specified by lpVerb.
lpParameters
    Additional parameters to use to perform the action. This would typically be additional command-line options to use, especially when running an executable file.
lpDirectory
    The path name of the working directory to use. If this is not specified, the current directory is used.
nShow
    One of the following flags specifying how to display any window that might be opened:
    SW_HIDE
        Hide the opened window.
    SW_MAXIMIZE
        Maximize the opened window.
    SW_MINIMIZE
        Minimize the opened window.
    SW_RESTORE
        Restore the opened window (not maximized nor minimized).
    SW_SHOW
        Show the opened window.
    SW_SHOWMAXIMIZED
        Show the opened window maximized.
    SW_SHOWMINIMIZED
        Show the opened window minimized.
Show the opened window minimized.

**SW_SHOWMINNOACTIVE**
Show the opened window minimized but do not activate the it.

**SW_SHOWNA**
Show the opened window in its current state but do not activate it.

**SW_SHOWNOACTIVATE**
Show the opened window in its most recent size and position but do not activate it.

**SW_SHOWNORMAL**
Show the opened window and activate it (as usual).

## Constant Definitions

- `Const ERROR_BAD_FORMAT = 11`
- `Const SE_ERR_ACCESSDENIED = 5`
- `Const SE_ERR_ASSOCINCOMPLETE = 27`
- `Const SE_ERR_DDEBUSY = 30`
- `Const SE_ERR_DDEFAIL = 29`
- `Const SE_ERR_DDETIMEOUT = 28`
- `Const SE_ERR_DLLNOTFOUND = 32`
- `Const SE_ERR_FNF = 2`
- `Const SE_ERR_NOASSOC = 31`
- `Const SE_ERR_OOM = 8`
- `Const SE_ERR_PNF = 3`
- `Const SE_ERR_SHARE = 26`
- `Const SW_HIDE = 0`
- `Const SW_MAXIMIZE = 3`
- `Const SW_MINIMIZE = 6`
- `Const SW_RESTORE = 9`
- `Const SW_SHOW = 5`
- `Const SW_SHOWMAXIMIZED = 3`
- `Const SW_SHOWMINIMIZED = 2`
- `Const SW_SHOWMINNOACTIVE = 7`
- `Const SW_SHOWNA = 8`
- `Const SW_SHOWNOACTIVATE = 4`
- `Const SW_SHOWNORMAL = 1`

## Example

Use **ShellExecute** to perform the following tasks automatically:

1. Run the program "C:\MyProg\startup.exe" with the "-fast" command line, in a maximized window.
2. Open the file "C:\Project\nucleus.doc" with its associated program, in a restored window.
3. Print the file "C:\Project\picture.bmp" via its associated program.

The example begins each task immediately and then finishes -- it does not wait for any of these three tasks to complete. The example runs when button Command1 is clicked. Naturally, to run this example, you must create a command button named Command1 on form window Form1.
'This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function ShellExecute Lib "shell32.dll" Alias "ShellExecuteA" (ByVal hwnd As Long, _
    ByVal lpOperation As String, ByVal lpFile As String, ByVal lpParameters As String, ByVal _
    lpDirectory As String, ByVal nShowCmd As Long) As Long
Public Const SW_MAXIMIZE = 3
Public Const SW_MINIMIZE = 6
Public Const SW_RESTORE = 9

Place the following code inside window Form1. ***

Private Sub Command1_Click()
    Dim retval As Long  ' return value
    ' 1. Run the program:
    retval = ShellExecute(Form1.hWnd, "open", "C:\MyProg\startup.exe", ",-fast", 
        "C:\MyProg\", _
        SW_MAXIMIZE)
    ' 2. Open the document:
    retval = ShellExecute(Form1.hWnd, "open", "C:\Project\nucleus.doc", ",", 
        "C:\Project\", _
        SW_RESTORE)
    ' 3. Print the document (minimized in case a window opens):
    retval = ShellExecute(Form1.hWnd, "print", "C:\Project\picture.bmp", ",", 
        "C:\Project\", _
        SW_MINIMIZE)
End Sub

See Also

ShellExecuteEx

Category

Shell

Go back to the Function listing.
Go back to the Reference section index.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com  Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shellexecute.html
ShellExecuteEx Function

Declare Function ShellExecuteEx Lib "shell32.dll" Alias "ShellExecuteExA"
(lpExecInfo As SHELLEXECUTEINFO) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Required Windows NT 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

ShellExecuteEx opens, prints, or executes a file using the Windows shell. When working with a non-executable file, the file is opened using its associated program. ShellExecuteEx can also open Windows Explorer windows. The function returns immediately after opening the file, starting the program, or performing whatever other action was specified.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns zero (use GetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

lpExecInfo

Information describing the action to perform through the Windows shell. After the function call completes, this structure also receives information about the function's result.

Example

Open the file "C:\Docs\readme.txt" using whatever program is associated with *.txt files (by default, Notepad). Wait until the user has closed the window before continuing with the example. The example begins when the user clicks button Command1.
Obviously, to use this example, you need to place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SHELLEXECUTEINFO
    cbSize As Long
    fMask As Long
    hwnd As Long
    lpVerb As String
    lpFile As String
    lpParameters As String
    lpDirectory As String
    nShow As Long
    hInstApp As Long
    lpIDList As Long
    lpClass As String
    hkeyClass As Long
    dwHotKey As Long
    hIcon As Long
    hProcess As Long
End Type
Public Const SEE_MASK_NOCLOSEPROCESS = &H40
Public Const SW_SHOWNORMAL = 1
Public Declare Function ShellExecuteEx Lib "shell32.dll" Alias "ShellExecuteExA" 
    (lpExecInfo As SHELLEXECUTEINFO) As Long
Public Const SE_ERR_FNF = 2
Public Const SE_ERR_NOASSOC = 31
Public Declare Function WaitForSingleObject Lib "kernel32.dll" (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long
Public Const INFINITE = &HFFFF
Public Const WAIT_TIMEOUT = &H102

' *** Place the following code inside window Form1. ***
Private Sub Command1_Click()
    Dim sei As SHELLEXECUTEINFO ' structure used by the function
    Dim retval As Long ' return value

    ' Load the information needed to open C:\Docs\readme.txt
    ' into the structure.
    With sei
        ' Size of the structure
        .cbSize = Len sei
        ' Use the optional hProcess element of the structure.
        .fMask = SEE_MASK_NOCLOSEPROCESS
        ' Handle to the window calling this function.
        .hwnd = Form1.hWnd
        ' The action to perform: open the file.
        .lpVerb = "open"
        .lpFile = "C:\Docs\readme.txt"
        .lpParameters = ""
.lpVerb = "open"
' The file to open.
.lpFile = "C:\Docs\readme.txt"
' No additional parameters are needed here.
.lpParameters = ""
' The default directory -- not really necessary in this case.
.lpDirectory = "C:\Docs\"
' Simply display the window.
.nShow = SW_SHOWNORMAL
' The other elements of the structure are either not used
' or will be set when the function returns.
End With

' Open the file using its associated program.
retval = ShellExecuteEx(sei)
If retval = 0 Then
' The function failed, so report the error.  Err.LastDllError
' could also be used instead, if you wish.
Select Case sei.hInstApp
Case SE_ERR_FNF
    Debug.Print "The file C:\Docs\readme.txt was not found."
Case SE_NOASSOC
    Debug.Print "No program is associated with *.txt files."
Case Else
    Debug.Print "An unexpected error occured."
End Select
Else
' Wait for the opened process to close before continuing.  Instead
' of waiting once for a time of INFINITE, this example repeatedly
checks to see if the
' is still open.  This allows the DoEvents VB function to be called,
preventing
' our program from appearing to lock up while it waits.
Do
    DoEvents
    retval = WaitForSingleObject(sei.hProcess, 0)
Loop While retval = WAIT_TIMEOUT
Debug.Print "Notepad (or whatever program was opened) has just
closed."
End If
End Sub

See Also
ShellExecute

Category
Shell
SHEmptyRecycleBin Function

Declare Function SHEmptyRecycleBin Lib "shell32.dll" Alias "SHEmptyRecycleBinA" (ByVal hwnd As Long, ByVal pszRootPath As String, ByVal dwFlags As Long) As Long

Platforms

- **Windows 95:** Requires Internet Explorer 4.0 or later with integrated shell installed.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 4.0 or later and Internet Explorer 4.0 or later with integrated shell installed.
- **Windows 2000:** Supported.
- **Windows CE:** Not Supported.

Description & Usage

SHEmptyRecycleBin deletes the contents of the Recycle Bin, displaying dialog boxes as desired. The function can empty the Recycle Bin of a particular drive, or it can empty all Recycle Bins as a whole.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

 hwnd

A **handle** to the window calling the function. This window will be the owner of any dialog boxes the function uses.
pszRootPath
A path belonging to the drive to empty the Recycle Bin of (this doesn't have to be the root path). To
empty all Recycle Bins as a whole, set this to an empty string.

dwFlags
A combination of zero or more of the following flags specifying additional options:
SHERB_NOCONFIRMATION
    Do not display a confirmation prompt for the user.
SHERB_NOPROGRESSUI
    Do not display the dialog box displaying the progress of emptying the Recycle Bin.
SHERB_NOSOUND
    Do not play the Empty Recycle Bin sound after the contents of the Recycle Bin are deleted.

Constant Definitions

Const SHERB_NOCONFIRMATION = &H1
Const SHERB_NOPROGRESSUI = &H2
Const SHERB_NOSOUND = &H4

Example

' This code is licensed according to the terms and conditions listed here.

' Delete the contents in the system's Recycle Bin, without
' showing the progress dialog. If an error occurs, be safe and
' make sure the proper Recycle Bin icon is used.
Dim retval As Long  ' return value

' Delete the contents of the system's Recycle Bin, if the user OKs it.
retval = SHEmptyRecycleBin(Form1.hWnd, "", SHERB_NOPROGRESSUI)

' If an error occured, be overly save and refresh the Recycle Bin
' icon. This probably isn't necessary, however.
If retval <> 0 Then  ' error
    retval = SHUpdateRecycleBinIcon()
End If

See Also

SHQueryRecycleBin, SHUpdateRecycleBinIcon

Category
She

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shemptyrecyclebin.html
SHFileOperation Function

Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" (lpFileOp As Byte) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

**SHFileOperation** copies, moves, renames, or deletes an object in the file system. Instead of performing the action silently as regular file API functions do, **SHFileOperation** uses the dialog box prompts of the shell. This function is also the proper way to send one or more files to the Recycle Bin instead of deleting them outright.

Return Value

If an error occured, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

Although *lpFileOp* is technically a **SHFILEOPSTRUCT** structure passed to **SHFileOperation**, a byte array must be used. This is because of a byte alignment quirk in Visual Basic that results in misplacing some data in the structure. Note the code used in the example to fix the problem. For more information about the reason for this, consult the **SHFILEOPSTRUCT** structure's page.

Parameters

*lpFileOp*

A **SHFILEOPSTRUCT** structure, copied properly into a byte array, that specifies the file operation to perform. It may also receive some feedback from the function.

Example

When the user clicks button cmdDelete, send all the files inside C:\Stuff\ to the Recycle Bin. Prompt the user as necessary. To
use this example, place a command button named cmdDelete on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SHFILEOPSTRUCT
    hwnd As Long
    wFunc As Long
    pFrom As String
    pTo As String
    fFlags As Integer
    fAnyOperationsAborted As Long
    hNameMappings As Long
    lpszProgressTitle As String
End Type
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" (lpFileOp As Byte) As Long
Public Const FO_DELETE = &H3
Public Const FOF_ALLOWUNDO = &H40
Public Const FOF_FILESONLY = &H80

' *** Place the following code inside the form window. ***

Private Sub cmdDelete_Click()
    Dim fos As SHFILEOPSTRUCT ' structure to pass to the function
    Dim sa(1 To 32) As Byte ' byte array to make structure properly sized
    Dim retval As Long ' return value

    ' Load the proper parameters into the structure.
    With fos
        ' The window invoking the file operation.
        .hwnd = Form1.hWnd
        ' Delete the specified files.
        .wFunc = FO_DELETE
        ' The list of files to delete.
        .pFrom = "C:\Stuff\*." & vbNullChar & vbNullChar
        ' Target information isn't needed for deletion of files.
        .pTo = vbNullChar & vbNullChar
        ' Allow Undo (i.e., send to Recycle Bin on delete), and do not delete subfolders.
        .fFlags = FOF_ALLOWUNDO Or FOF_FILESONLY
        ' The rest of the structure isn't needed for this example.
        .fAnyOperationsAborted = 0
        .hNameMappings = 0
        .lpszProgressTitle = vbNullChar
    End With

    ' Transfer the contents of the structure object into the byte array in order to compensate for a byte alignment problem.
' Send those files to the Recycle Bin.
retval = SHFileOperation(sa(1))

' Although not necessary for this example, transfer the (possibly changed) 
' byte array back into the structure. This would be necessary in case 
SHFileOperation might
' have placed some feedback information into the structure passed to it.
CopyMemory sa(21), sa(19), 12
CopyMemory fos, sa(1), Len(fos)

End Sub

Category

Shell

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shfileoperation.html
SHFreeNameMappings Function

Declare Sub SHFreeNameMappings Lib "shell32.dll" (ByVal hNameMappings As Long)

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

SHFreeNameMappings frees a file mapping object created by the SHFileOperation function. The file mapping object consists of an array of SHNAMEMAPPING structures identifying the renamed files.

Return Value

SHFreeNameMappings does not return a value.

Visual Basic-Specific Issues

NOTE: Although I know how to create and free a file mapping object, I still have not been able to figure out how to actually access its contents. If you know how to do this, please e-mail me.

Parameters

hNameMappings

A handle to the file mapping object to free.

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SHFILEOPSTRUCT
    hwnd As Long
wFunc As Long
pFrom As String
pTo As String
fFlags As Integer
fAnyOperationsAborted As Long
hNameMappings As Long
lpszProgressTitle As String
End Type
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" _
    (Destination As Any, Source As Any, ByVal Length As Long)
Public Declare Function SHFileOperation Lib "shell32.dll" Alias "SHFileOperationA" _
    (lpFileOp As Byte) As Long
Public Declare Sub SHFreeNameMappings Lib "shell32.dll" (ByVal hNameMappings As Long)
Public Const FO_COPY = &H2
Public Const FOF_ALLOWUNDO = &H40
Public Const FOF_FILESONLY = &H80
Public Const FOF_WANTMAPPINGHANDLE = &H20
'
' Copy the files "C:\Folder\*.\*" to the folder "C:\NewFolder\".
Dim fos As SHFILEOPSTRUCT ' structure to pass to the function
Dim s(1 To 24) As Byte     ' byte array to make structure properly sized
Dim retval As Long         ' return value
'
' Load the proper parameters into the structure.
With fos
    ' The window invoking the file operation.
    hwnd = Form1.hWnd
    ' Copy the specified files.
    wFunc = FO_COPY
    ' The list of source files to copy.
    pFrom = "C:\Folder\*.\*" & vbNullChar & vbNullChar
    ' The path to copy the files to.
    pTo = "C:\NewFolder\" & vbNullChar & vbNullChar
    ' Allow Undo, and do not copy subfolders.
    ' Also ask for a file mapping object.
    fFlags = FOF_ALLOWUNDO Or FOF_FILESONLY Or FOF_WANTMAPPINGHANDLE
    ' The rest of the structure isn't needed for this example.
    fAnyOperationsAborted = 0
    hNameMappings = 0
    lpszProgressTitle = vbNullChar
End With
'
' Transfer the contents of the structure object into the byte
' array in order to compensate for a byte alignment problem.
CopyMemory sa(1), fos, LenB(fos)
CopyMemory sa(19), structarray(21), 12
'
' Send those files to the Recycle Bin.
retval = SHFileOperation(sa(1))
'
' Transfer the byte array back into the structure. This is necessary
' because the mapping handle has been added to the structure.
CopyMemory  sa(21), sa(19), 12
CopyMemory  fos, sa(1), Len(fos)

' If you knew how to access the file mapping object, it would be done here.

' Free the file mapping object to free up resources.
SHFreeNameMappings  fos.hNameMappings

Category

Shell

Back to the Function list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shfreenamemappings.html
SHGetFileInfo Function

Declare Function SHGetFileInfo Lib "shell32.dll" Alias "SHGetFileInfoA" (ByVal pszPath As Any, ByVal dwFileAttributes As Long, psfi As SHFILEINFO, ByVal cbFileInfo As Long, ByVal uFlags As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

SHGetFileInfo retrieves information about a file system object in the shell. Such file system objects include files, directories, virtual folders, and drives. The information retrieved generally relates to how the object appears in the system shell.

Return Value

If the uFlags parameter contains the SHGFI_EXETYPE flag, the function returns a value identifying the type of the executable file. If both the low-order and high-order words are not zero, the file is a Windows application. If only the low-order word is not zero but the high-order word is zero, the file is an MS-DOS based program or batch file or a 32-bit Windows console application. If the entire return value is zero, an error occurred.

If the uFlags parameter instead contains the SHGFI_SYSICONINDEX flag, the function returns a handle to an internal image list which contains the icon requested by the function. If this value is zero, an error occurred.

If neither of those two flags are specified, the return value is a non-zero value if successful, or zero if an error occurred.

Visual Basic-Specific Issues

None.
Parameters

pszPath
The name of the file, directory, or drive to retrieve information about. If uFlags contains the SHGFI_PIDL flag, this is instead a pointer to an ITEMIDLIST structure (a PIDL) identifying a file system object. If uFlags contains the SHGFI_USEFILEATTRIBUTES flag, this parameter does not have to refer to an actual file; the function will instead retrieve attributes which the file would have if it existed (such as information relating to its extension).

dwFileAttributes
If uFlags contains the SHGFI_USEFILEATTRIBUTES flag, this parameter is a combination of the following flags specifying the file attributes of the theoretical file. If that flag is not specified, this parameter is ignored.

FILE_ATTRIBUTE_ARCHIVE
An archive file (which most files are).

FILE_ATTRIBUTE_COMPRESSED
A file residing in a compressed drive or directory.

FILE_ATTRIBUTE_DIRECTORY
A directory instead of a file.

FILE_ATTRIBUTE_HIDDEN
A hidden file, not normally visible to the user.

FILE_ATTRIBUTE_NORMAL
An attribute-less file (cannot be combined with other attributes).

FILE_ATTRIBUTE_READONLY
A read-only file.

FILE_ATTRIBUTE_SYSTEM
A system file, used exclusively by the operating system.

psfi
The structure that receives information about the file.

cbFileInfo
The size in bytes of the structure passed as psfi.

uFlags
A combination of the following flags specifying what information to retrieve about the file system object:

SHGFI_ATTRIBUTES
Retrieve the attributes of the file system object and put that information into the structure.

SHGFI_DISPLAYNAME
Retrieve the display name of the file system object. This is its full name as presented in the system shell.

SHGFI_FILE_ATTRIBUTE
Retrieve what type of executable file was passed to the function. The result is returned by the function. This flag cannot be used with any other flags.

SHGFI_ICON
Retrieve a handle to the icon used to depict the file system object and put it into the structure. After using the handle, your program must use DestroyIcon to free up resources.

SHGFI_ICONLOCATION
Retrieve the name of the file which holds the icon used to depict the file system object and put it into the structure.

SHGFI_LARGEICON
When used with SHGFI_ICON, causes the function to retrieve the file system object's large icon.
SHGFI_LINKOVERLAY
When used with SHGFI_ICON, causes the function to add the link overlay (the "shortcut arrow" in the bottom right corner) to the icon.

SHGFI_OPENICON
When used with SHGFI_ICON, causes the function to use the icon which depicts the file system object as being open. (This usually applies only to icons used to depict folders or directories.)

SHGFI_PIDL
Indicates that pszPath contains a PIDL instead of a filename.

SHGFI_SELECTED
When used with SHGFI_ICON, causes the function to add the tint of the system highlight color to the file system object's icon (making it look "selected").

SHGFI_SHELLICONSIZE
When used with SHGFI_ICON, causes the function to retrieve a shell-sized icon instead of the standard size of icon.

SHGFI_SMALLICON
When used with SHGFI_ICON, causes the function to retrieve the file system object's small icon.

SHGFI_SYSICONINDEX
Retrieve the index of the icon in an internal image list and place it into the structure. The function returns a handle to the image list itself. Note that trying to access any icon with an index not equal to the one placed inside the structure will cause unexpected results!

SHGFI_TYPENAME
Retrieve the name of the file type of the file system object and place it into the structure.

SHGFI_USEFILEATTRIBUTES
Instead of specifying a file that actually exists, pszPath gives the name of a possibly fictitious file and dwFileAttributes identifies the pretend file's attributes. The function then behaves as if the file did exist and retrieves the appropriate information. This flag cannot be used with the SHGFI_ATTRIBUTES, SHGFI_EXETYPE, or SHGFI_PIDL flags.

Constant Definitions

Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_COMPRESSED = &H800
Const FILE_ATTRIBUTE_DIRECTORY = &H10
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H0
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4
Const SHGFI_ATTRIBUTES = &H800
Const SHGFI_DISPLAYNAME = &H200
Const SHGFI_EXETYPE = &H2000
Const SHGFI_ICON = &H100
Const SHGFI_ICONLOCATION = &H1000
Const SHGFI_LARGEICON = &H0
Const SHGFI_LINKOVERLAY = &H8000
Const SHGFI_OPENICON = &H2
Const SHGFI_PIDL = &H8
Const SHGFI_SELECTED = &H10000
Example

' This code is licensed according to the terms and conditions listed here.

' Retrieve information about a generic MP3 file found in the system shell. The file specified by the function does not actually exist, but its properties will reflect what any MP3 file would probably have (the .mp3 extension and the "archive" attribute).
Dim info As SHFILEINFO  ' receives information about the file
Dim retval As Long  ' return value of the function

' Retrieve information about what the C:\dummy.mp3 file would look like if it existed.
retval = SHGetFileInfo("C:\dummy.mp3", FILE_ATTRIBUTE_ARCHIVE, info, Len(info), SHGFI_USEFILEATTRIBUTES Or SHGFI_TYPENAME Or SHGFI_ICON)

' Display the name of the .mp3 file type on the computer.
' (Note how the trailing nulls are removed from the string.)
Debug.Print "The file type of .mp3 files is "; Left(info.szTypeName, InStr(info.szTypeName, vbNullChar) - 1)

' Draw the icon used for MP3 files in the corner of window Form1.
retval = DrawIcon(Form1.hDC, 0, 0, info.hIcon)
' Destroy the icon handle to save resources.
retval = DestroyIcon(info.hIcon)

Category

Shell

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: December 22, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shgetfileinfo.html
SHGetFolderLocation Function

Declare Function SHGetFolderLocation Lib "shell32.dll" Alias "SHGetFolderLocationA"
(ByVal hwndOwner As Long, ByVal nFolder As Long, ByVal hToken As Long, ByVal
dwReserved As Long, ppidl As Long) As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Not Supported.
- Windows NT: Not Supported
- Windows CE: Not Supported.

Note: SHGetFolderLocation may be available on platforms other than Windows 2000. It also appears in shfolder.dll, a redistributable DLL. The Declare for that function is identical to the one above, except for the filename. shfolder.dll is not necessarily guaranteed to be on a system.

Description & Usage

SHGetFolderLocation creates a pointer to an ITEMIDLIST structure (a.k.a. a PIDL) referring to a special folder on the computer. The PIDL can refer to special folders which are either a physical path on a drive or a virtual folder. After your program is finished using the PIDL obtained by this function, use CoTaskMemFree to release it.

Return Value

The function returns one of the following flags:

S_OK
The function completed successfully.
S_FALSE
The CSIDL of the special folder is valid but is a virtual folder.
E_INVALIDARG
The CSIDL is not valid.

Visual Basic-Specific Issues

None.

Parameters
hwndOwner

A handle to the window calling the function. This window will own any dialog boxes created by this function.

nFolder

The CSIDL of the special folder to get a PIDL to.

hToken

A token identifying the user. This is normally 0, but may be other values for security purposes.

dwReserved

Reserved -- set to 0.

ppidl

Receives a PIDL to the desired special folder.

Example

' This code is licensed according to the terms and conditions listed here.

' Open the Browse for Folder dialog box and display both the display name and
' the actual name of the folder (if it is not a virtual folder). Any folder under My
' Computer
' may be selected.
Dim bi As BROWSEINFO  ' structure passed to the function
Dim pidl As Long  ' PIDL to the user's selection
Dim physpath As String  ' string used to temporarily hold the physical path
Dim retval As Long  ' return value

' Initialize the structure to be passed to the function.
bi.hwndOwner = Form1.hWnd  ' window Form1 is the owner of the dialog box
' Specify the My Computer virtual folder as the root
retval = SHGetFolderLocation(Form1.hWnd, CSIDL_DRIVES, 0, 0, bi.pidlRoot)
' Make room in the buffer to get the [virtual] folder's display name
bi.pszDisplayName = Space(260)
bi.lpszTitle = "Please choose a folder."  ' Message displayed to the user
bi.ulFlags = 0  ' no flags are needed here
bi.lpfn = 0  ' no callback function is being used
bi.lParam = 0  ' not needed
bi.iImage = 0  ' this will be set by the function

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
   ' Remove the empty space from the display name variable.
   bi.pszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
   Debug.Print "The user selected: "; bi.pszDisplayName
   ' If the folder is not a virtual folder, display its physical location.
   physpath = Space(260)  ' make room in the buffer
   retval = SHGetPathFromIDList(pidl, physpath)
   If retval = 0 Then
      Debug.Print "Physical Location: (virtual folder)"
Else
    ' Remove the empty space and display the result.
    physpath = Left(physpath, InStr(physpath, vbNullChar) - 1)
    Debug.Print "Physical Location: "; physpath
End If
' Free the pidl returned by the function.
CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to
' identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

See Also

SHGetFolderPath, SHGetSpecialFolderLocation

Category

Shell

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shgetfolderlocation.html
SHGetPathFromIDList Function

Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA"
(ByVal pidl As Long, ByVal pszPath As String) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Not Supported.

Description & Usage

SHGetPathFromIDList determines the path of the folder referred to by a pointer to an ITEMIDLIST structure (a.k.a. a PIDL). The function only works if the PIDL refers to a physical directory on the system; it cannot get the name of a virtual folder.

Return Value

If an error occurred or the PIDL refers to a virtual folder, the function returns 0. If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **pidl**: The PIDL to determine the physical path of, if it is not a virtual folder.
- **pszPath**: Receives a null-terminated string holding the path which the PIDL identifies. The string passed initially to the function must be at least 260 characters long.

Example

' This code is licensed according to the terms and conditions listed here.
Open the Browse for Folder dialog box and display both the display name and the actual name of the folder (if it is not a virtual folder). Any folder under My Computer may be selected.

Dim bi As BROWSEINFO  ' structure passed to the function
Dim pidl As Long  ' PIDL to the user's selection
Dim physpath As String  ' string used to temporarily hold the physical path
Dim retval As Long  ' return value

' Initialize the structure to be passed to the function.
bi.hwndOwner = Form1.hWnd  ' window Form1 is the owner of the dialog box
' Specify the My Computer virtual folder as the root
retval = SHGetSpecialFolderLocation(Form1.hWnd, CSIDL_DRIVES, bi.pidlRoot)
' Make room in the buffer to get the [virtual] folder's display name
bi.pszDisplayName = Space(260)
bi.lpszTitle = "Please choose a folder."  ' Message displayed to the user
bi.ulFlags = 0  ' no flags are needed here
bi.lpfn = 0  ' no callback function is being used
bi.lParam = 0  ' not needed
bi.iImage = 0  ' this will be set by the function

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
' Remove the empty space from the display name variable.
b.ipszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
Debug.Print "The user selected: "; bi.pszDisplayName
' If the folder is not a virtual folder, display its physical location.
physpath = Space(260)  ' make room in the buffer
retval = SHGetPathFromIDList(pidl, physpath)
If retval = 0 Then
    Debug.Print "Physical Location: (virtual folder)"
Else
    ' Remove the empty space and display the result.
    physpath = Left(physpath, InStr(physpath, vbNullChar) - 1)
    Debug.Print "Physical Location: "; physpath
End If
' Free the pidl returned by the function.
CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

Category

Shell
SHGetFolderPath Function

Declare Function SHGetFolderPath Lib "shell32.dll" Alias "SHGetFolderPathA" (ByVal hwndOwner As Long, ByVal nFolder As Long, ByVal hToken As Long, ByVal dwFlags As Long, ByVal lpszPath As String) As Long

Platforms

- Windows 95: Not Supported.
- Windows 98: Not Supported.
- Windows NT: Not Supported.
- Windows CE: Not Supported.

Note: SHGetFolderPath may be available on platforms other than Windows 2000. It also appears in shfolder.dll, a redistributable DLL. The Declare for that function is identical to the one above, except for the filename. shfolder.dll is not necessarily guaranteed to be on a system.

Description & Usage

SHGetFolderPath retrieves the name of the path of a special folder on the system. This function only works with special folders which are physical directories on a disk. The function will fail if the special folder is a virtual folder.

Return Value

The function returns one of the following flags:

S_OK
   The function completed successfully.
S_FALSE
   The CSIDL of the special folder is valid but is a virtual folder.
E_INVALIDARG
   The CSIDL is not valid.

Visual Basic-Specific Issues

None.

Parameters
hwndOwner

A handle to the window calling the function, which will own any dialog boxes the function may create.

nIndex

The CSIDL of the special folder to get the path of.

hToken

A token identifying the user. This is normally 0, but may be other values for security purposes.

dwFlags

One of the following flags specifying which path to retrieve:

- SHGFP_TYPE_CURRENT
  - Retrieve the folder's current path.
- SHGFP_TYPE_DEFAULT
  - Retrieve the folder's default path.

lpszPath

Receives a null-terminated string holding the path name of the special folder. A string at least 260 characters long must be passed as this parameter to receive the result.

Constant Definitions

Const S_OK = &H0
Const S_FALSE = &H1
Const E_INVALIDARG = &H80070057
Const SHGFP_TYPE_CURRENT = 0
Const SHGFP_TYPE_DEFAULT = 1

Example

' This code is licensed according to the terms and conditions listed here.

' Display the path of the directory used as the My Documents special folder. This might not be "C:\My Documents" if the user renamed it!
Dim pathname As String  ' receives the path of My Documents
Dim retval As Long  ' return value

' Make enough room in the buffer to receive the string.
pathname = Space(260)
' Get the path name of the My Documents special folder
retval = SHGetFolderPath(Form1.hWnd, CSIDL_PERSONAL, 0, SHGFP_TYPE_CURRENT, pathname)
' Remove the empty space from the string.
pathname = Left(pathname, InStr(pathname, vbNullChar) - 1)
' Display the result.
Debug.Print "The My Documents special folder is: "; pathname

See Also

SHGetFolderLocation, SHGetSpecialFolderPath

Category
SHGetSpecialFolderLocation Function

Declare Function SHGetSpecialFolderLocation Lib "shell32.dll" (ByVal hwndOwner As Long, ByVal nFolder As Long, ppidl As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows 2000: Supported but Obsolete; use SHGetFolderLocation instead.
- Windows CE: Not Supported.

Description & Usage

SHGetSpecialFolderLocation creates a pointer to an ITEMIDLIST structure (a.k.a. a PIDL) refering to a special folder on the computer. The PIDL can refer to special folders which are either a physical path on a drive or a virtual folder. After your program is finished using the PIDL obtained by this function, use CoTaskMemFree to release it.

Return Value

If successful, the function returns zero. If an error occured, the function returns an error code.

Visual Basic-Specific Issues

None.

Parameters

hwndOwner
A handle to the window calling the function. This window will own any dialog boxes created by this function.

nFolder
The CSIDL of the special folder to get a PIDL to.

ppidl
Receives a PIDL to the desired special folder.

Example

Open the Browse for Folder dialog box and display both the display name and the actual name of the folder (unless it is a
virtual folder). Any folder under My Computer may be selected. To run this example, place a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SHGetSpecialFolderLocation Lib "shell32.dll" (ByVal hwndOwner As Long, ByVal nFolder As Long, ppidl As Long) As Long
Public Const CSIDL_DRIVES = &H11
Public Type BROWSEINFO
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type
Public Declare Function SHBrowseForFolder Lib "shell32.dll" Alias "SHBrowseForFolderA" (lpbi As BROWSEINFO) As Long
Public Declare Function SHGetPathFromIDList Lib "shell32.dll" Alias "SHGetPathFromIDListA" (ByVal pidl As Long, ByVal pszPath As String) As Long
Public Declare Sub CoTaskMemFree Lib "ole32.dll" (ByVal pv As Long)

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim bi As BROWSEINFO ' structure passed to the function
    Dim pidl As Long ' PIDL to the user's selection
    Dim physpath As String ' string used to temporarily hold the physical path
    Dim retval As Long ' return value

    ' Initialize the structure to be passed to the function.
    With bi
        .hwndOwner = Me.hWnd
        ' Specify the My Computer virtual folder as the root.
        retval = SHGetSpecialFolderLocation(Me.hWnd, CSIDL_DRIVES, .pidlRoot)
        ' Make room in the buffer to get the [virtual] folder's display name.
        .pszDisplayName = Space(260)
        ' Message displayed to the user.
        .lpszTitle = "Please choose a folder."
        ' Nothing else needs to be set.
        .ulFlags = 0
        .lpfn = 0
        .lParam = 0
        .iImage = 0
    End With
    ' Do something with the selected folder...
End Private
End With

' Open the Browse for Folder dialog box.
pidl = SHBrowseForFolder(bi)
' If the user selected something, display its display name
' and its physical location on the system.
If pidl <> 0 Then
    ' Remove the empty space from the display name variable.
    bi.pszDisplayName = Left(bi.pszDisplayName, InStr(bi.pszDisplayName, vbNullChar) - 1)
    Debug.Print "The user selected: "; bi.pszDisplayName
    phypath = Space(260)
    retval = SHGetPathFromIDList(pidl, phypath)
    If retval = 0 Then
        Debug.Print "Physical Location: (virtual folder)"
    Else
        ' Remove the empty space and display the result.
        phypath = Left(phypath, InStr(phypath, vbNullChar) - 1)
        Debug.Print "Physical Location: "; phypath
    End If
' Free the pidl returned by the function.
CoTaskMemFree pidl
End If

' Whether successful or not, free the PIDL which was used to
' identify the My Computer virtual folder.
CoTaskMemFree bi.pidlRoot

See Also

SHGetFolderLocation, SHGetSpecialFolderPath

Category

Shell

Go back to the Function listing.
Go back to the Reference section index.

---

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shgetspecialfolderlocation.html
SHGetSpecialFolderPath Function

Declare Function SHGetSpecialFolderPath Lib "shell32.dll" Alias "SHGetSpecialFolderPathA" (ByVal hwndOwner As Long, ByVal lpszPath As String, ByVal nFolder As Long, ByVal fCreate As Long) As Long

Platforms

- **Windows 95:** Requires Internet Explorer 4.0 or later.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 4.0 or later with Internet Explorer 4.0 or later.
- **Windows 2000:** Supported but Obsolete; use SHGetFolderPath instead.
- **Windows CE:** Not Supported.

Description & Usage

**SHGetSpecialFolderPath** retrieves the name of the path of a special folder on the system. This function only works with special folders which are physical directories on a disk. The function will fail if the special folder is a virtual folder.

Return Value

If an error occurred, the function returns 0. If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- **hwndOwner**
  A handle to the window calling the function, which will own any dialog boxes the function may create.
- **lpszPath**
  Receives a null-terminated string holding the path name of the special folder. A string at least 260 characters long must be passed as this parameter to receive the result.
- **nIndex**
The CSIDL of the special folder to get the path of.

fCreate
If 0, do not create the special folder if it does not yet exist. If a non-zero value, create the special folder if it does not yet exist.

Example

' This code is licensed according to the terms and conditions listed here.

' Display the path of the directory used as the My Documents special folder. This might not be "C:\My Documents" if the user renamed it!
Dim pathname As String  ' receives the path of My Documents
Dim retval As Long  ' return value

' Make enough room in the buffer to receive the string.
pathname = Space(260)  ' Get the path name of the My Documents special folder
retval = SHGetSpecialFolderPath(Form1.hWnd, pathname, CSIDL_PERSONAL, 0)  
' Remove the empty space from the string.
pathname = Left(pathname, InStr(pathname, vbNullChar) - 1)  
' Display the result.
Debug.Print "The My Documents special folder is: "; pathname

See Also

SHGetFolderPath, SHGetSpecialFolderLocation

Category

Shell

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shgetspecialfolderpath.html
ShowCursor Function

Declare Function ShowCursor Lib "user32.dll" (ByVal bShow As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

ShowCursor either shows or hides the mouse cursor. This is not done directly, but rather by
incrementing or decrementing a counter. Each function call raises or lowers the counter by 1. If the
counter is negative, the cursor is invisible. If it non-negative, the cursor is visible. The function returns
the value of the counter after changing it.

bShow

If zero, decrement the counter by 1. If non-zero, increment the counter by 1.

Example:

' Hide the mouse cursor for 5 seconds.
Dim counter As Long  ' receives value of cursor visibility counter

' Hide the cursor by decrementing the counter until it is negative
Do
    counter = ShowCursor(0)  ' decrement by 1
Loop Until counter < 0  ' keep looping until cursor is hidden

Sleep 5000  ' pause execution for 5 seconds (5000 milliseconds)

' Show the cursor by incrementing the counter until it is not negative
Do
    counter = ShowCursor(1)  ' increment by 1
Loop Until counter => 0  ' keep looping until cursor is visible

Category: Cursor

Go back to the alphabetical Function listing.
ShowWindow Function

Declare Function ShowWindow Lib "user32.dll" (ByVal hwnd As Long, ByVal nCmdShow As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

ShowWindow shows (or hides) a window in a certain manner. For example, the function can minimize, maximize, or restore a given window. The function returns 0 if the window had been hidden before the call, or a non-zero value if it had been visible.

hwnd
The handle of the window to change the show status of.

nCmdShow
Exactly one of the following flags specifying how to show the window:

- **SW_HIDE = 0**
  Hide the window.
- **SW_MAXIMIZE = 3**
  Maximize the window.
- **SW_MINIMIZE = 6**
  Minimize the window.
- **SW_RESTORE = 9**
  Restore the window (not maximized nor minimized).
- **SW_SHOW = 5**
  Show the window.
- **SW_SHOWMAXIMIZED = 3**
  Show the window maximized.
- **SW_SHOWMINIMIZED = 2**
  Show the window minimized.
- **SW_SHOWMINNOACTIVE = 7**
  Show the window minimized but do not activate it.
- **SW_SHOWNA = 8**
  Show the window in its current state but do not activate it.
- **SW_SHOWNOACTIVATE = 4**
  Show the window in its most recent size and position but do not activate it.
- **SW_SHOWNORMAL = 1**
  Show the window and activate it (as usual).

Example:
Maximize the window Form1. Before doing so, make sure that the window is visible.

```vbnet
Dim retval As Long ' return value
retval = ShowWindow(Form1.hWnd, SW_HIDE) ' display the window if it's hidden
retval = ShowWindow(Form1.hWnd, SW_MAXIMIZE) ' maximize the window
```

See Also: IsIconic, IsZoomed
Category: Windows
SHQueryRecycleBin Function

Declare Function SHQueryRecycleBin Lib "shell32.dll" Alias "SHQueryRecycleBinA"
(ByVal pszRootPath As String, pSHQueryRBInfo As SHQUERYRBINFO) As Long

Platforms

- **Windows 95**: Requires Internet Explorer 4.0 or later with integrated shell installed.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later and Internet Explorer 4.0 or later with integrated shell installed.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

SHQueryRecycleBin retrieves information about how many files (or other items) are currently in the Recycle Bin as well as how much disk space they consume. This function will work with the Recycle Bin specific to a certain drive, as well as work with the Recycle Bin as considered over the entire system.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

- **pszRootPath**
  A path which is on the drive to query the Recycle Bin of. If this string is empty, the function instead queries all Recycle Bins on the system as a whole.

- **pSHQueryRBInfo**
  Receives the number of bytes in the Recycle Bin and the number of items in it. The cbSize member of the structure must be initialized before calling the function.

Example
This code is licensed according to the terms and conditions listed [here](#).

Display the number of items in the Recycle Bin on the C: drive and the size of it.

```vbnet
Dim rbinfo As SHQUERYRBIINFO ' information about the bin
Dim retval As Long ' return value

' Initialize the size of the structure.
rbinfo.cbSize = Len(rbinfo)
' Query the contents of C:'s Recycle Bin.
retval = SHQueryRecycleBin("C:\", rbinfo) ' the path doesn't have to be the root path
' Display the number of items in the Recycle Bin, if the value is within Visual Basic's numeric display limits.
If (rbinfo.i64NumItems.LowPart And &H80000000) = &H80000000 Or rbinfo.i64NumItems.HighPart > 0 Then
    Debug.Print "Recycle Bin contains more than 2,147,483,647 items."
Else
    Debug.Print "Recycle Bin contains"; rbinfo.i64NumItems.LowPart; "items."
End If
' Likewise display the number of bytes the Recycle Bin is taking up.
If (rbinfo.i64Size.LowPart And &H80000000) = &H80000000 Or rbinfo.i64Size.HighPart > 0 Then
    Debug.Print "Recycle Bin consumes more than 2,147,483,647 bytes."
Else
    Debug.Print "Recycle Bin consumes"; rbinfo.i64Size.LowPart; "bytes."
End If
```

### See Also

*SHEmptyRecycleBin*

### Category

*Shell*

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

**Last Modified:** August 31, 1999

This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/s/shqueryrecyclebin.html](http://www.vbapi.com/ref/s/shqueryrecyclebin.html)
SHUpdateRecycleBinIcon Function

Declare Function SHUpdateRecycleBinIcon Lib "shell32.dll" () As Long

Platforms

- **Windows 95**: Requires Internet Explorer 4.0 with integrated shell installed.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 4.0 or later and Internet Explorer 4.0 or later with integrated shell installed.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

`SHUpdateRecycleBinIcon` updates the Recycle Bin icon on the desktop to reflect the state of the systemwide Recycle Bin. Since the other Recycle Bin management functions will update this icon on their own, there's almost no reason why your applications would need to call this function explicitly.

Return Value

If an error occurred, the function returns a non-zero error code. If successful, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

None.

Example
Delete the contents in the system's Recycle Bin, without showing the progress dialog. If an error occurs, be safe and make sure the proper Recycle Bin icon is used.

Dim retval As Long  ' return value

Delete the contents of the system's Recycle Bin, if the user OKs it.
retval = SHEmptyRecycleBin(Form1.hWnd, "", SHERB_NOPROGRESSUI)

If an error occurred, be overly save and refresh the Recycle Bin icon. This probably isn't necessary, however.
If retval <> 0 Then  ' error
    retval = SHUpdateRecycleBinIcon()
End If

See Also

SHEmptyRecycleBin

Category

Shell

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: September 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shupdaterecyclebinicon.html
Sleep Function

Declare Sub Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)

Platforms: Win 32s, Win 95/98, Win NT

Sleep pauses program execution for a certain amount of time. This is more accurate than using a do-nothing loop, waiting for a certain amount of time to pass. The function does not return a value.

_dwMilliseconds_

The number of milliseconds to halt program execution for.

Example:

' Pause the program for 2 seconds, displaying the system
' time before and after the pause.

Debug.Print "The time is "; Time$ ' display the current time

Sleep 2000 ' 2000 milliseconds = 2 seconds to delay

Debug.Print "The time is "; Time$ ' this time will be 2 seconds later

Category: Other

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/sleep.html
sndPlaySound Function

Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA" (ByVal lpszSoundName As String, ByVal uFlags As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

NOTE: The function sndPlaySound is obsolete. It is superseded by the PlaySound function.

sndPlaySound plays either a .WAV file or a system-defined sound. If the SND_NODEFAULT flag is used, the function returns 0 if the .WAV file (or system sound) cannot be found and 1 if it is. If the flag is not set, the function always returns 1 and plays the Windows Default sound if the specified sound cannot be found.

lpszSoundName
Either the path and filename of the .WAV file to play, or the name of the system sound to play.

uFlags
Zero or more of the following flags specifying how to play the sound:

SND_ALIAS = &H10000
Play a Windows sound (such as SystemStart, Asterisk, etc.).

SND_ASYNC = &H1
Continue program execution immediately after starting to play the sound.

SND_FILENAME = &H20000
Play the specified filename.

SND_LOOP = &H8
Play the sound repeatedly until sndPlaySound is called again with lpszSoundName = "". SND_ASYNC must also be set.

SND_NODEFAULT = &H2
Do not play the Windows default sound if the specified sound cannot be found.

SND_NOSTOP = &H10
Do not stop playing any currently playing sounds.

SND_NOWAIT = &H2000
Do not wait if the sound driver is busy.

SND_SYNC = &H0
Wait until the sound has finished playing before continuing program execution.

Example:

' Play the Empty Recycle Bin system sound and pause
Dim retval As Long

retval = sndPlaySound("EmptyRecycleBin", SND_ALIAS Or SND_SYNC) ' play the associated sound

See Also: PlaySound
Category: Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/sndplaysound.html
socket Function

Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, ByVal protocol As Long) As Long

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

`socket` creates a new socket used for network communications. The socket is assigned a particular communications protocol (such as TCP/IP) upon creation. Before using this function, Winsock must be initialized via calling `WSAStartup`.

Return Value

If successful, the function returns a descriptor of the socket that was created. If an error occurred, the function returns INVALID_SOCKET (use `WSAGetLastError` to get the error code).

Visual Basic-Specific Issues

None.

Parameters

- **af**
  
  The address family of the protocol to use with the socket. For the Internet, this will be AF_INET (for IP).

- **prototype**
  
  The type of protocol to use with the socket. This may be one of the following values:
  
  - **SOCK_DGRAM**
    
    A datagram-based protocol. For the Internet, this will typically be UDP/IP.

  - **SOCK_STREAM**
    
    A two-way stream-based protocol. For the Internet, this will typically be TCP/IP.

- **protocol**
  
  The protocol to use with the socket. For the Internet, set this to zero.
Constant Definitions

Const INVALID_SOCKET = &HFFFFFFFF

Example

Download the main page of this web site (http://www.vbapi.com). This example supports a very crude implementation of HyperText Transport Protocol (HTTP), sending a request to the server and receiving the document. The document downloaded, with HTTP headers removed, is output to the Debug window. To prevent the program from appearing to lock up in the event of a momentary interruption in the transfer, a nonblocking socket is used. To use this example, place a command button named cmdDownload on a form window.

Note the careful use of GoTo in this example. Since there are lots of things that can go wrong, and WSACleanup must be called at the end no matter what happens, the GoTo statements skip down to the end if an unrecoverable error occurs. If VB had better exception handling, I would use that instead of GoTo.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Const AF_INET = 2
Public Const SOCK_STREAM = 1
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Type hostent
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Declare Function htons Lib "wsock32.dll" (ByVal hostshort As Integer) As Integer
Public Declare Function socket Lib "wsock32.dll" (ByVal af As Long, ByVal prototype As Long, _
    ByVal protocol As Long) As Long
Public Type sockaddr
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type

Public Declare Function connect Lib "wsock32.dll" (ByVal s As Long, name As sockaddr, ByVal namelen As Long) As Long
Public Declare Function ioctlsocket Lib "wsock32.dll" (ByVal s As Long, ByVal cmd As Long, ByVal argp As Long) As Long
Public Const FIONBIO = &H8004667E
Public Declare Function send Lib "wsock32.dll" (ByVal s As Long, ByVal buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function recv Lib "wsock32.dll" (ByVal s As Long, ByVal buf As Any, ByVal length As Long, ByVal flags As Long) As Long
Public Declare Function closesocket Lib "wsock32.dll" (ByVal s As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Const SOCKET_ERROR = -1

' Define a useful macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private Sub cmdDownload_Click()
    Dim wsockinfo As WSADATA ' info about Winsock
    Dim sock As Long ' the socket descriptor
    Dim pHostinfo As Long ' pointer to info about the host computer
    Dim hostinfo As hostent ' info about the host computer
    Dim pIPAddress As Long ' pointer to host's IP address
    Dim ipAddress As Long ' host's IP address
    Dim sockinfo As sockaddr ' settings for the socket
    Dim buffer As String ' buffer for sending and receiving data
    Dim reply As String ' accumulates server's reply
    Dim retval As Long ' generic return value

    ' Begin a Winsock session.
    retval = WSASStartup(MAKEWORD(2, 2), wsockinfo)
    If retval <> 0 Then
        Debug.Print "Unable to initialize Winsock! --"; retval
        Exit Sub
    End If
End Sub
'Get information about the server to connect to.
pHostinfo = gethostbyname("www.vbapi.com")
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve host!"
    GoTo Cleanup
End If
'Copy information about the server into the structure.
CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
If hostinfo.h_addrtype <> AF_INET Then
    Debug.Print "Couldn't get IP address of www.vbapi.com!"
    GoTo Cleanup
End If
'Get the server's IP address out of the structure.
CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
CopyMemory ipAddress, ByVal pIPAddress, 4
'
Create a socket.
sock = socket(AF_INET, SOCK_STREAM, 0)
If sock = SOCKET_ERROR Then
    Debug.Print "Unable to create socket!"
    GoTo Cleanup
End If
'Make a connection to www.vbapi.com:80 (where the web server listens).
With sockinfo
    'Use Internet Protocol (IP)
    .sin_family = AF_INET
    'Connect to port 80.
    .sin_port = htons(80)
    'Connect to this IP address.
    .sin_addr = ipAddress
    'Padding characters.
    .sin_zero = String(8, vbNullChar)
End With
Debug.Print "Attempting to connect...."
retval = connect(sock, sockinfo, Len(sockinfo))
If retval <> 0 Then
    Debug.Print "Unable to connect!"
    GoTo Cleanup
End If
'Send an HTTP/GET request for the /index.html file.
buffer = "GET / HTTP/1.1" & vbCrLf & _
    "Host: www.vbapi.com" & vbCrLf & _
    "User-Agent: HTTP-Test-Program" & vbCrLf & vbCrLf
retval = send(sock, ByVal buffer, Len(buffer), 0)
Debug.Print "Sent request.  Waiting for reply..."
'Make the socket non-blocking, so calls to recv don't halt the program
waiting for input.
retval = ioctlsocket(sock, FIONBIO, 1)
' Read the response from the other system. A more sophisticated program
' would watch to see if the connection ever times out (i.e., if the
connection is
' lost). For brevity, such code is omitted here.
Do
    buffer = Space(4096)
    retval = recv(sock, ByVal buffer, Len(buffer), 0)
    If retval <> 0 And retval <> SOCKET_ERROR Then
        reply = reply & Left(buffer, retval)
    End If
' Process background events so the program doesn't appear to freeze.
    DoEvents
Loop Until retval = 0

' Print the response from the server.
    Debug.Print "Document Retrieved:"
    Debug.Print reply

' Perform the necessary cleanup at the end.
Cleanup:
    retval = closesocket(sock)
    retval = WSACleanup()
End Sub

See Also
closesocket

Category
Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/socket.html
StartDoc Function

Declare Function StartDoc Lib "gdi32.dll" Alias "StartDocA" (ByVal hdc As Long, lpdi As DOCINFO) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

StartDoc initiates a job for a printer. The function prepares the print spooler to receive information about the document to print. The StartDoc and EndDoc functions must bracket all the code which draws the document on the printer (see the example for an illustration).

Return Value

If an error occurred, the function returns either a zero or negative value (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns the print job identifier of the document to print.

Visual Basic-Specific Issues

None.

Parameters

- hdc
  - A handle to a device context to the printer to print the document on.
- lpdi
  - Briefly describes the document to print.

Example

' This code is licensed according to the terms and conditions listed here.
' Print out a page with an ellipse drawn with a thickened black pen on it. The page is printed on the computer's default printer.

' The following are special declarations needed to allow string manipulation functions to use pointers to strings.
Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

' Variable declarations
Dim hPrintDC As Long  ' handle to the printer's device context
Dim di As DOCINFO  ' information about the document to print
Dim hPen As Long  ' handle to the pen to draw the ellipse with
Dim hOldPen As Long  ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long  ' 3076-byte buffer
Dim pi2 As PRINTER_INFO_2  ' receives info about the default printer
Dim printret As Long  ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long  ' receives space requires for EnumPrinters
Dim retval As Long  ' return value

' Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.
retval = EnumPrinters(PRINTER_ENUM_DEFAULT, "", 2, buffer(0), 3076, spaceneeded, printret)
If retval = 0 Then
    Debug.Print "No default printer is configured."
    End  ' abort the program
End If

' Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.
pi2.pPrinterName = Space(lstrlen(buffer(1)))
retval = lstrcpy(pi2.pPrinterName, buffer(1))
pi2.pDriverName = Space(lstrlen(buffer(4)))
retval = lstrcpy(pi2.pDriverName, buffer(4))

' Create a device context to the printer, using its default initialization.
hPrintDC = CreateDC("", pi2.pPrinterName, 0, ByVal CLng(0))
' Create a solid black brush with a thickness of 5.
hPen = CreatePen(PS_SOLID, 5, RGB(0, 0, 0))

' Load information about the document to print into the structure.
di.cbSize = Len(di)  ' size of structure
di.lpszDocName = "Printer API Demonstration"  ' name of document
di.lpszOutput = 0  ' do not print to a file
di.lpszDatatype = ""  ' data type of file doesn't apply
di-fwType = 0  ' no additional information

' Begin the print job.
retval = StartDoc(hPrintDC, di)
' Begin the first and only page to print.
retval = StartPage(hPrintDC)
' Select the pen for use with the printer.
hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

EndDoc

Category

Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 1, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/startdoc.html
StartPage Function

Declare Function StartPage Lib "gdi32.dll" (ByVal hDC As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 2.0 or later.

Description & Usage

StartPage informs the print spooler that the program is about to begin drawing the contents of a printed page. The StartPage and EndPage functions must bracket all code which draws the contents of the printed page. The print spooler must have already been informed that it is receiving a document to be printed via StartDoc. Windows 95/98: This function automatically resets the printer configuration to its settings when the device context to the printer was first obtained. Any alterations to the prior settings must be made after each time StartPage is called.

Return Value

If an error occurred, the function returns either zero or a negative value (Windows NT, 2000: use GetLastError to get the error code). If successful, the function returns a positive value.

Visual Basic-Specific Issues

None.

Parameters

hDC

A handle to a device context to the printer being used to print the document.

Example

' This code is licensed according to the terms and conditions listed here.

' Print out a page with an ellipse drawn with a thickened black
The following are special declarations needed to allow string manipulation functions to use pointers to strings.

Declare Function `lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As String, ByVal lpString2 As Long) As Long
Declare Function `lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Long) As Long

Variable declarations

Dim hPrintDC As Long ' handle to the printer's device context
Dim di As DOCINFO ' information about the document to print
Dim hPen As Long ' handle to the pen to draw the ellipse with
Dim hOldPen As Long ' handle to the printer's previously selected pen
Dim buffer(0 To 3076 / 4) As Long ' 3076-byte buffer
Dim pi2 As PRINTER_INFO_2 ' receives info about the default printer
Dim printret As Long ' receives the number of printers returned from EnumPrinters
Dim spaceneeded As Long ' receives space requires for EnumPrinters
Dim retval As Long ' return value

Get the device and driver names of the default printer. For a more detailed description of the semi-confusing code below, consult the EnumPrinters page.

retval = EnumPrinters(PRINTER_ENUM_DEFAULT, "", 2, buffer(0), 3076, spaceneeded, printret)
If retval = 0 Then
    Debug.Print "No default printer is configured."
    End ' abort the program
End If

Copy the device and driver names to the structure. All the other information retrieved is not needed and is omitted here.

pi2.pPrinterName = Space(lstrlen(buffer(1)))
retval = lstrcpy(pi2.pPrinterName, buffer(1))
pi2.pDriverName = Space(lstrlen(buffer(4)))
retval = lstrcpy(pi2.pDriverName, buffer(4))

Create a device context to the printer, using its default initialization.
hPrintDC = CreateDC("", pi2.pPrinterName, 0, ByVal CLng(0))

Create a solid black brush with a thickness of 5.
hPen = CreatePen(PS_SOLID, 5, RGB(0, 0, 0))

Load information about the document to print into the structure.
di.cbSize = Len(di) ' size of structure
di.lpszDocName = "Printer API Demonstration" ' name of document
di.lpszOutput = 0 ' do not print to a file
di.lpszDatatype = "" ' data type of file doesn't apply
di/fwType = 0 ' no additional information

Begin the print job.
retval = StartDoc(hPrintDC, di)

Begin the first and only page to print.
retval = StartPage(hPrintDC)
' Select the pen for use with the printer.
hOldPen = SelectObject(hPrintDC, hPen)
' Draw an ellipse with bounding rectangle corners (1000,1500)-(2000,3000)
retval = Ellipse(hPrintDC, 1000, 1500, 2000, 3000)
' Restore the printer's previously selected pen.
retval = SelectObject(hPrintDC, hOldPen)
' End information about the first and only page.
retval = EndPage(hPrintDC)
' End information about the document.
retval = EndDoc(hPrintDC)
' The printer will now begin printing the document.

' Delete the pen created for drawing.
retval = DeleteObject(hPen)
' Delete the device context to the printer.
retval = DeleteDC(hPrintDC)

See Also

EndPage

Category

Printers

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: November 5, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/startpage.html
StretchBlt Function

Declare Function StretchBlt Lib "gdi32.dll" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, ByVal xSrc As Long, ByVal ySrc As Long, ByVal hSrcWidth As Long, ByVal nSrcHeight As Long, ByVal dwRop As Long) As Long

StretchBlt copies a section of an image from one device to another. This function also allows you to change the original size and dimensions of the image section, unlike the related function BitBlt. In addition to using the straight "copy" method, you can specify other ways of copying the image with the dwRop parameter. What the function actually does is perform a binary operation on the color of the source and destination pixel to calculate the color of the pixel in the transferred image. The point you specify as the location of the copied image in the target object will be the upper-left corner of the image portion. The function returns 0 if the function failed and 1 if it succeeded.

hDestDC
The device context of the target object (the one that receives the image piece).
x
The x coordinate of the point to put the image inside the target.
y
The y coordinate of the point to put the image inside the target.
nWidth
The width of the image piece in the target.
nHeight
The height of the image piece in the target.
xSrc
The x coordinate of the upper-left corner of the image piece in the source.
ySrc
The y coordinate of the upper-left corner of the image piece in the source.
nSrcWidth
The width of the image piece in the source.
nSrcHeight
The height of the image piece in the source.
dwRop
Exactly one of the following flags specifying what method to use to copy the source image:
SRCAND = &H8800C6
Logically And the two color values (destination = source And destination).
SRCCOPY = &HCC0020
Copy the source image exactly (destination = source).
SRCERASE = &H440328
Logically And the source image and the destination's binary inverse (destination = source And (Not destination).
SRCINVERT = &H660046
   Logically Xor the two color values (destination = source Xor destination).
SRCPAINT = &HEE0086
   Logically Or the two color values (destination = source Or destination).

Example:

' Copy a portion of the image in PictureBox1 to PictureBox2 and
' stretch it to triple its original width. The image's original dimensions are
' 16x32; its new ones are 48x32.
' Source image coordinates: (45,50)-(60,81)
' Target image coordinates: (0,0)-(47,31)

   Dim retval As Long  ' function return value
   
   retval = BitBlt(PictureBox2.hdc, 0, 0, 48, 32, PictureBox1.hdc, 45, 50, 16, 32, SRCCOPY)

See Also:  BitBlt
Category:  Bitmaps

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SubtractRect Function

Declare Function SubtractRect Lib "user32.dll" (lprcDst As RECT, lprcSrc1 As RECT, lprcSrc2 As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

SubtractRect subtracts a smaller rectangle from a larger one. Rectangle subtraction is defined as follows. The large and small rectangles must intersect completely along one entire side, neither extending farther along that side than the other. In other words, they must share a common side. If this is true, then all of the large rectangle that is not also part of the small rectangle is also a rectangle. This rectangle is the subtraction rectangle. This rectangle is put into the variable passed as lprcDst. If the two rectangles fail to meet the criteria, lprcDst is set equal to the large rectangle and the function returns 0. If subtraction is possible, the function returns 1.

lprcDst
Variable that receives the subtraction rectangle.

lprcSrc1
The large rectangle; that is, the rectangle subtracted from.

lprcSrc2
The small rectangle; that is, the rectangle subtracted.

Example:

' A demonstration of rectangle subtract. target = big - small.
' big = (20,30)-(70,80). small = (20,30)-(40,80). Note that the left side of the two rectangles is
' common. target will be set to (41,30)-(70,80).
Dim target As RECT, big As RECT, small As RECT
Dim retval As Long ' return value

' Set the big and small rectangles
retval = SetRect(big, 20, 30, 70, 80) ' big = (20,30)-(70,80)
retval = SetRect(small, 20, 30, 40, 80) ' small = (20,30)-(40,80)

' Subtract small from big and put the result into target
retval = SubtractRect(target, big, small) ' now target = (41,30)-(70,80)

See Also: IntersectRect, UnionRect
Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SwapMouseButton Function

Declare Function SwapMouseButton Lib "user32.dll" (ByVal bSwap As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SwapMouseButton alters the mapping used for the mouse buttons. Windows allows you to interpret left-button clicks as right-button clicks and vice versa. This functionality is designed for people who use the mouse left-handed. Of course, Windows is normally the only program to ever change this, so do it with caution -- it of course affects mouse usage in all programs as well as Windows itself. The function returns 1 if successful, or 0 if an error occurred.

\*bSwap\*

Determines the mouse button mapping. 0 means do not swap the buttons (left means left, right means right). 1 means swap the left and right buttons (left means right, right means left). Note that "swapping" is relative to the normal left-left mapping, not to the current one.

Example:

' Swap the left and right mouse buttons. You probably would never
' do this in a program unless the user specifically asked for it (as he/she does in
' Windows' Control Panel).
Dim retval As Long ' return value

retval = SwapMouseButton(1) ' switch the left and right buttons
' Now left means right even if this had previously been the case.
Debug.Print "The mouse buttons are currently swapped."

Category: Mouse

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/swapmousebutton.html
SystemParametersInfo Function

Declare Function SystemParametersInfo Lib "user32.dll" Alias "SystemParametersInfoA" (ByVal uAction As Long, ByVal uiParam As Long, pvParam As Any, ByVal fWinIni As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

SystemParametersInfo reads or sets information about numerous settings in Windows. These include Windows's accessibility features as well as various settings for other things. The exact behavior of the function depends on the flag passed as uAction. All sizes and dimensions used by this function are measured in pixels. Whenever using a structure to receive information, the member identifying the size of the structure (if one exists) must be correctly set beforehand. Note that, when setting properties via this function, that the user may not expect such settings to be changed; normally only the Control Panel applets set many of these properties. The return value in almost all cases (there are exceptions, which are noted) is 0 if the function failed, or a non-zero value if the function succeeded. Note to Visual Basic users: Whenever passing a string as pvParam, must explicitly add the ByVal keyword before it. This must also be done when explicitly setting the pvParam parameter to 0. See the examples for a demonstration of when and how this is done.

uAction

Exactly one of the following flags specifying the action taken by the function:

SPI_GETACCESSTIMEOUT = 60

Retrieve information about the time-out period associated with the accessibility features. uiParam must be the size of the ACCESSTIMEOUT structure. pvParam is an ACCESSTIMEOUT structure receiving the time-out settings.

SPI_GETANIMATION = 72

Win 95/98 only: Retrieve information about the animation effects associated with the user's actions. uiParam must be 0. pvParam is an ANIMATIONINFO structure receiving the animation effects settings.

SPI_GETBEEP = 1

Determine if the warning beeper is on or off. uiParam must be 0. pvParam is a Long-type variable which receives 0 if the warning beeper is off, or a non-zero value if it is on.

SPI_GETBORDER = 5

Retrieve the window sizing border multiplier factor, which determines the width of a window's sizing border. uiParam must be 0. pvParam is a Long-type variable which receives the current setting.

SPI_GETDEFAULTINPUTLANG = 89

Win 95/98 only: Retrieve a handle to the keyboard layout used for the system's default input language. uiParam must be 0. pvParam is a Long-type variable which receives the handle.

SPI_GETDRAGFULLWINDOWS = 38

Win 95/98 only: Determine if Windows displays the entire contents of a window when it is moved or resized (instead of merely displaying an outline of it). uiParam must be 0. pvParam is a Long-type variable which receives 0 if the contents are not displayed, or a non-zero value if they are.

SPI_GETFASTTASKSWITCH = 35

Determine if fast Alt-Tab task switching is enabled. uiParam must be 0. pvParam is a Long-type variable which receives 0 if fast task switching is not enabled, or a non-zero value if it is.
SPI_GETFILTERKEYS = 50
Retrieve the settings of the FilterKeys accessibility feature. *uiParam* must be the size of the FILTERKEYS structure. *pvParam* is a FILTERKEYS structure which receives the current settings of FilterKeys.

SPI_GETFONTSMOOTHING = 74
Determine whether font smoothing is enabled or not. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if font smoothing is not enabled, or a non-zero value if it is.

SPI_GETGRIDGRANULARITY = 18
Retrieve the current granularity of the desktop sizing grid. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the current setting.

SPI_GETHIGHCONTRAST = 66
*Win 95/98 only:* Retrieve the settings of the HighContrast accessibility feature. *uiParam* must be 0. *pvParam* is a HIGHCONTRAST structure which receives the current settings of HighContrast.

SPI_GETICONMETRICS = 45
*Win 95/98 only:* Retrieve the metrics associated with icons, which determine how Windows displays icons. *uiParam* must be 0. *pvParam* is an ICONMETRICS structure which receives the icon metrics.

SPI_GETICONTITLELOGFONT = 31
Retrieve information about the logical font used to display the titles of icons. *uiParam* must be the size of the LOGFONT structure. *pvParam* is a LOGFONT structure which receives information about the logical font.

SPI_GETICONTITLEWRAP = 25
Determine if Windows word-wraps the text of icon titles. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if word-wrapping is not enabled, or a non-zero value if it is.

SPI_GETKEYBOARDDELAY = 22
Retrieve the current keyboard repeat delay setting, which is the time before a held key begins to repeat. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the current delay setting (a value between 0 and 3).

SPI_GETKEYBOARDPREF = 68
*Win 95/98 only:* Determine if the user relies on the keyboard instead of the mouse and wants programs to display keyboard interfaces which are otherwise hidden. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if the user does not rely on the keyboard, or a non-zero value if the user does.

SPI_GETKEYBOARDSPEED = 10
Retrieve the current keyboard repeat speed setting, which is the speed between repeats when a key is held. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the current speed setting (a value between 0 and 31).

SPI_GETLOWPOWERACTIVE = 83
*Win 95/98 only:* Determine if the system enters a low-power mode after a period of inactivity. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if low-power mode is not enabled, or a non-zero value if it is.

SPI_GETLOWPOWERTIMEOUT = 79
*Win 95/98 only:* Retrieve the time, in seconds, which must elapse before Windows enters low-power mode. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the current timeout value.

SPI_GETMENUDROPALIGNMENT = 27
Determine if popup menus are left- or right-aligned. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if the menus are right-aligned, or a non-zero value if they are left-aligned.

SPI_GETMINIMIZEDMETRICS = 43
*Win 95/98 only:* Retrieve the metrics associated with minimized windows, which specify how Windows displays minimized windows. *uiParam* must be the size of the MINIMIZEDMETRICS structure. *pvParam* is a MINIMIZEDMETRICS structure which receives the minimized window metrics.

SPI_GETMOUSE = 3
Retrieve the x-axis and y-axis threshold values for the mouse as well as the mouse speed. *uiParam* must be 0. *pvParam* is a 3-element array of Long-type variables which receives the x-threshold, y-threshold, and mouse...
SPI_GETMOUSEKEYS = 54
Retrieve the settings of the MouseKeys accessibility feature. *uiParam* must be the size of the *MOUSEKEYS* structure. *pvParam* is a *MOUSEKEYS* structure which receives the current settings of MouseKeys.

SPI_GETMOUSETRAILS = 94
*Win 95/98 only:* Retrieve the current mouse trails setting. *uiParam* must be 0. *pvParam* receives 0 or 1 if mouse trails are not enabled, or a value greater than one identifying the number of cursor images making up the mouse trail.

SPI_GETNONCLIENTMETRICS = 41
*Win 95/98 only:* Retrieve the metrics associated with the nonclient areas of windows, which determine how Windows renders nonclient areas. *uiParam* must be 0. *pvParam* is a *NONCLIENTMETRICS* structure which receives the nonclient area metrics.

SPI_GETPOWEROFFACTIVE = 84
*Win 95/98 only:* Determine if the system enters a power-off mode after a period of inactivity. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if power-off mode is not enabled, or a non-zero value if power-off mode is enabled.

SPI_GETPOWEROFFTIMEOUT = 80
*Win 95/98 only:* Retrieve the time-out value, in seconds, that must elapse before Windows enters power-off mode. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the time-out value for power-off mode.

SPI_GETSCREENREADER = 70
*Win 95/98 only:* Determine if a screen reader utility is running. If it is, applications should present more textual output instead of graphical output to help the reader function better. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if no screen reader is running, or a non-zero value if one is.

SPI_GETSCREENSAVETIMEOUT = 14
Retrieve the time-out period, in seconds, which must elapse before the screen saver begins running. *uiParam* must be 0. *pvParam* is a Long-type variable which receives the time-out value.

SPI_GETSERIALKEYS = 62
*Win 95/98 only:* Retrieve the settings of the SerialKeys accessibility feature. *uiParam* must be 0. *pvParam* is a *SERIALKEYS* structure which receives the current settings of SerialKeys.

SPI_GETSHOWSOUNDS = 56
Determine if the user desires visual information to replace or supplement otherwise audio-only output. *uiParam* must be 0. *pvParam* is a Long-type variable which receives 0 if programs should not display this added visual information, or a non-zero value if they should.

SPI_GETSOUNDSENTRY = 64
Retrieve the settings of the SoundSentry accessibility feature. *uiParam* must be the size of the *SOUNDSENTRY* structure. *pvParam* is a *SOUNDSENTRY* structure which receives the current settings of SoundSentry.

SPI_GETSTICKYKEYS = 58
Retrieve the settings of the StickyKeys accessibility feature. *uiParam* must be the size of the *STICKYKEYS* structure. *pvParam* is a *STICKYKEYS* structure which receives the settings of StickyKeys.

SPI_GETTOGGLEKEYS = 52
Retrieve the settings of the ToggleKeys accessibility feature. *uiParam* must be the size of the *TOGGLEKEYS* structure. *pvParam* is a *TOGGLEKEYS* structure which receives the current settings of ToggleKeys.

SPI_GETWINDOWSEXTENSION = 92
*Win 95/98 only:* Determine if the Windows extensions are installed. Win 95 requires that Microsoft Plus! be
installed to get the extensions; the extensions are integrated as a part of Win 98. $uiParam$ must be 1. $pvParam$ must be 0. The function returns 0 if the extensions are disabled, or a non-zero value if they are enabled.

$SPI\_GET\_WORK\_AREA = 48$

**Win 95/98 only:** Retrieve the size of the working area, which is a rectangle identifying the area of the desktop not obscured by the taskbar. $uiParam$ must be 0. $pvParam$ is a $RECT$ structure which receives the rectangle of the working area.

$SPI\_ICON\_HORIZONTAL\_SPACING = 13$

Set the width of the icon spacing cell. $uiParam$ is the new setting for the width. $pvParam$ must be 0.

$SPI\_ICON\_VERTICAL\_SPACING = 24$

Set the height of the icon spacing cell. $uiParam$ is the new setting for the height. $pvParam$ must be 0.

$SPI\_LANG\_DRIVER = 12$

**Win 95/98 only:** Retrieve the filename of the language driver. $uiParam$ must be 0. $pvParam$ is a String (with sufficient buffer space) which receives the filename of the language driver.

$SPI\_SET\_ACCESS\_TIME\_OUT = 61$

Set information associated with the time-out period associated with the accessibility features. $uiParam$ must be the size of the $ACCESS\_TIME\_OUT$ structure. $pvParam$ is an $ACCESS\_TIME\_OUT$ structure holding the new time-out period settings.

$SPI\_SET\_ANIMATION = 73$

**Win 95/98 only:** Set information about the animation effects associated with the user's actions. $uiParam$ must be 0. $pvParam$ is an $ANIMATION\_INFO$ structure holding the new animation effects settings.

$SPI\_SET\_BEEP = 2$

Turn the warning beeper on or off. $uiParam$ is 0 to turn the beeper off, or a non-zero value to turn it on. $pvParam$ must be 0.

$SPI\_SET\_BORDER = 6$

Set the window sizing border multiplier factor, which determines the width of a window's sizing border. $uiParam$ is the new setting. $pvParam$ must be 0.

$SPI\_SET\_CUR\_SORS = 87$

Reload the images used for the system cursors. $uiParam$ and $pvParam$ must be 0.

$SPI\_SET\_DEFAULT\_INPUT\_LANG = 90$

**Win 95/98 only:** Set the keyboard layout used for the system's default input language. $uiParam$ is a handle to the keyboard layout to use. $pvParam$ must be 0.

$SPI\_SET\_DESKTOP\_PATTERN = 21$

Set the current desktop pattern bitmap by causing Windows to reread the Pattern setting in the [Desktop] section of WIN.INI. Both $uiParam$ and $pvParam$ must be 0.

$SPI\_SET\_DESKTOP\_WALL\_PAPER = 20$

Set the current desktop wallpaper bitmap. $uiParam$ must be 0. $pvParam$ is a String holding the filename of the bitmap file to use as the wallpaper.

$SPI\_SET\_DOUBLE\_CLICK\_TIME = 32$

Set the time, in milliseconds, within which two successive mouse clicks must occur for Windows to interpret the input to be a double click. $uiParam$ is the new double click time. $pvParam$ must be 0.

$SPI\_SET\_DOUBLE\_CLICK\_HEIGHT = 30$

Set the height of the rectangle within which two successive mouse clicks must occur for Windows to interpret the input to be a double click. $uiParam$ is the new height. $pvParam$ must be 0.

$SPI\_SET\_DOUBLE\_CLICK\_WIDTH = 29$

Set the width of the rectangle within which two successive mouse clicks must occur for Windows to interpret the input to be a double click. $uiParam$ is the new width. $pvParam$ must be 0.

$SPI\_SET\_DRAG\_FULL\_WINDOWS = 37$

**Win 95/98 only:** Turn dragging of full windows (displaying the contents of a window while moving or resizing instead of just an empty border) on or off. $uiParam$ is 0 to turn dragging of full windows off, or a non-zero value to turn it on.
SPI_SETDRAGHEIGHT = 77
  **Win 95/98 only:** Set the height of the rectangle which the cursor must move out of with a button depressed for Windows to begin a drag operation. *uiParam* is the new height. *pvParam* must be 0.

SPI_SETDRAGWIDTH = 76
  **Win 95/98 only:** Set the width of the rectangle which the cursor must move out of with a button depressed for Windows to begin a drag operation. *uiParam* is the new width. *pvParam* must be 0.

SPI_SETFASTTASKSWITCH = 36
  Turn fast Alt-Tab task switching on or off. *uiParam* is 0 to turn fast switching off, or a non-zero value to turn it on. *pvParam* must be 0.

SPI_SETFILTERKEYS = 51
  Set the settings of the FilterKeys accessibility feature. *uiParam* must be the size of the FILTERKEYS structure. *pvParam* is 0.

SPI_SETFONTSMOOTHING = 75
  Turn font smoothing on or off. *uiParam* is 0 to turn font smoothing off, or a non-zero value to turn it on.

SPI_SETGRIDGRANULARITY = 19
  Set the granularity of the desktop sizing grid. *uiParam* is the new setting. *pvParam* must be 0.

SPI_SETHIGHCONTRAST = 67
  **Win 95/98 only:** Set the settings of the HighContrast accessibility feature. *uiParam* must be 0. *pvParam* is a HIGHCONTRAST structure holding the new settings for HighContrast.

SPI_SETICONMETRICS = 46
  **Win 95/98 only:** Set the metrics associated with icons, which specify how Windows displays icons. *uiParam* must be 0. *pvParam* is an ICONMETRICS structure holding the new icon metrics.

SPI_SETICONS = 88
  Reload the images used for the system icons. *uiParam* and *pvParam* must be 0.

SPI_SETICONTITLELOGFONT = 34
  Set the logical font used to display the text of icon titles. *uiParam* must be the size of the LOGFONT structure. *pvParam* is a LOGFONT structure holding the logical font to use.

SPI_SETICONTITLEWRAP = 26
  Turn the word-wrapping of icon titles on or off. *uiParam* is 0 to turn word-wrapping off, or a non-zero value to turn it on. *pvParam* must be 0.

SPI_SETKEYBOARDDELAY = 23
  Set the keyboard repeat-delay setting, which is the time that must elapse before a held key begins to repeat. *uiParam* is the new delay setting, between 0 and 3 inclusive. *pvParam* must be 0.

SPI_SETKEYBOARDPREF = 69
  **Win 95/98 only:** Tell Windows whether the user depends on the keyboard as the main input device and wishes programs to display additional keyboard-based interfaces. *uiParam* is 0 to not indicate a keyboard preference, or a non-zero value to indicate one. *pvParam* must be 0.

SPI_SETKEYBOARDSPEED = 11
  Set the keyboard repeat-speed setting, which specifies the rate at which keys are repeated. *uiParam* is the new speed setting, between 0 and 31 inclusive. *pvParam* must be 0.

SPI_SETLANGTOGGLE = 91
  **Win 95/98 only:** Set the hot key used to switch keyboard input languages. The new value is read from the registry at HKEY_CURRENT_USER\keyboard layout\toggle: 1 sets the hot key to ALT+SHIFT, 2 sets the hot key to CTRL+SHIFT, and 3 disables the hot key. *uiParam* and *pvParam* must be 0.

SPI_SETLOWPOWERACTIVE = 85
  **Win 95/98 only:** Turn the low-power mode after a period of inactivity on or off. *uiParam* is 0 to deactivate low-power mode, or 1 to activate it. *pvParam* must be 0.

SPI_SETLOWPOWERTIMEOUT = 81
  **Win 95/98 only:** Set the time-out period (in seconds) which must elapse after user input before Windows enters low-power mode. *uiParam* is the new time-out period. *pvParam* must be 0.
SPI_SETMENUDROPALIGNMENT = 28
Set whether pop-up menus open to the left or to the right. uiParam is 0 if they open to the left, or a non-zero value if they open to the right. pvParam must be 0.

SPI_SETMINIMIZEDMETRICS = 44
Win 95/98 only: Set the metrics associated with minimized windows, which determine how Windows displays minimized windows. uiParam must be 0. pvParam is a MINIMIZEDMETRICS structure holding the new metrics.

SPI_SETMOUSE = 4
Set the x-axis and y-axis threshold values for the mouse as well as the mouse speed. uiParam must be 0. pvParam is a three-element Long-type array; the first element is the x-threshold, the second element is the y-threshold, and the third element is the mouse speed.

SPI_SETMOUSEBUTTONSWAP = 33
Swap or unswap the meanings of the left and right mouse buttons. uiParam is 0 to restore the left-to-left/right-to-right mapping, or a non-zero value to swap the buttons into a left-to-right/right-to-left mapping. pvParam must be 0.

SPI_SETMOUSEKEYS = 55
Set the settings of the MouseKeys accessibility feature. uiParam must be the size of the MOUSEKEYS structure. pvParam is a MOUSEKEYS structure holding the new settings for MouseKeys.

SPI_SETMOUSETRAILS = 93
Win 95/98 only: Set the length of the mouse trail. uiParam is 0 or 1 to turn mouse trails off, or a value greater than one identifying the number of cursor images used to make up the mouse trail. pvParam must be 0.

SPI_SETNONCLIENTMETRICS = 42
Win 95/98 only: Set the metrics associated with the non-client areas of windows, which determine how Windows displays the non-client areas. uiParam must be 0. pvParam is a NONCLIENTMETRICS structure holding the new metrics.

SPI_SETPOWEROFFACTIVE = 86
Win 95/98 only: Set whether the system powers down after a period of inactivity. uiParam is 0 to not enter power-off mode, or a non-zero value to enter power-off mode. pvParam must be 0.

SPI_SETPOWEROFFTIMEOUT = 82
Win 95/98 only: Set the time-out period, in seconds, which must elapse before Windows enters power-off mode. uiParam is the new time-out period. pvParam must be 0.

SPI_SETSCREENREADER = 71
Win 95/98 only: Set whether a screen-reading program is currently running. uiParam is 0 if no reader is in use, or a non-zero value if one is. pvParam must be 0.

SPI_SETSCREENSAVETIMEOUT = 17
Set whether Windows activates a screen saver after a period of inactivity. uiParam is 0 to turn the screen saver off, or a non-zero value to turn it on. pvParam must be 0.

SPI_SETSCREENSAVETIMEOUT = 15
Set the time-out period, in seconds, which must elapse before Windows launches a screen saver. uiParam is the new time-out period. pvParam must be 0.

SPI_SETSERIALKEYS = 63
Win 95/98 only: Set the settings of the SerialKeys accessibility feature. uiParam must be 0. pvParam is a SERIALKEYS structure holding the new settings for SerialKeys.

SPI_SETPENWINDOWS = 49
Win 95/98 only: Load or unload Microsoft Pen for Windows, if available. uiParam is 0 to unload Pen, or a non-zero value to load Pen. pvParam must be 0.

SPI_SETPENWINDOWS = 49
Win 95/98 only: Load or unload Microsoft Pen for Windows, if available. uiParam is 0 to unload Pen, or a non-zero value to load Pen. pvParam must be 0.

SPI_SETPOWEROFFACTIVE = 86
Win 95/98 only: Set whether the system powers down after a period of inactivity. uiParam is 0 to not enter power-off mode, or a non-zero value to enter power-off mode. pvParam must be 0.

SPI_SETPOWEROFFTIMEOUT = 82
Win 95/98 only: Set the time-out period, in seconds, which must elapse without user input before Windows enters power-off mode. uiParam is the new time-out period. pvParam must be 0.

SPI_SETSCREENREADER = 71
Win 95/98 only: Set whether a screen-reading program is currently running. uiParam is 0 if no reader is in use, or a non-zero value if one is. pvParam must be 0.

SPI_SETSCREENSAVETIMEOUT = 17
Set whether Windows activates a screen saver after a period of inactivity. uiParam is 0 to turn the screen saver off, or a non-zero value to turn it on. pvParam must be 0.

SPI_SETSCREENSAVETIMEOUT = 15
Set the time-out period, in seconds, which must elapse before Windows launches a screen saver. uiParam is the new time-out period. pvParam must be 0.

SPI_SETSERIALKEYS = 63
Win 95/98 only: Set the settings of the SerialKeys accessibility feature. uiParam must be 0. pvParam is a SERIALKEYS structure holding the new settings for SerialKeys.

SPI_SETSHOWSOUNDS = 57
Turn the ShowSounds accessibility feature on or off. uiParam is 0 to turn ShowSounds off, or a non-zero value to turn ShowSounds on. pvParam must be 0.
SPI_SETSOUNDENTRY = 65
Set the settings of the SoundSentry accessibility feature. \textit{uiParam} must be the size of the \texttt{SOUNDENTRY} structure. \textit{pvParam} is a \texttt{SOUNDENTRY} structure holding the new settings for SoundSentry.

SPI_SETSTICKYKEYS = 59
Set the settings of the StickyKeys accessibility feature. \textit{uiParam} is the size of the \texttt{STICKYKEYS} structure. \textit{pvParam} is a \texttt{STICKYKEYS} structure holding the new settings for StickyKeys.

SPI_SETTOGGLEKEYS = 53
Set the settings of the ToggleKeys accessibility feature. \textit{uiParam} is the size of the \texttt{TOGGLEKEYS} structure. \textit{pvParam} is a \texttt{TOGGLEKEYS} structure holding the new settings for ToggleKeys.

SPI_SETWORKAREA = 47
\textbf{Win 95/98 only}: Set the rectangle defining the working area of the desktop. The working area is the area of the desktop not obscured by the taskbar. \textit{uiParam} must be 0. \textit{pvParam} is a \texttt{RECT} structure holding the new working area rectangle.

\textbf{\textit{uiParam}}
The purpose of this parameter varies with \textit{uAction}.

\textbf{\textit{pvParam}}
The purpose of this parameter varies with \textit{uAction}. In VB, if this is to be set as a string or to 0, the ByVal keyword must preceed it.

\textbf{\textit{fWinIni}}
Zero or more of the following flags specifying the change notification to take place. Generally, this can be set to 0 if the function merely queries information, but should be set to something if the function sets information.

SPIF_SENDWININICHANGE = &H2
Broadcast the change made by the function to all running programs.

SPIF_UPDATEINIFILE = &H1
Save the change made by the function to the user profile.

\textbf{Example #1:}

' Set the desktop wallpaper to the Clouds.bmp that comes with Windows.
' Note how the string passed to the function is preceeded by the ByVal keyword.
Dim windir As String  ' receives the path of the Windows directory
Dim cloupath As String  ' filename of Clouds.bmp
Dim retval As Long  ' return value

' Get the path of the Windows directory.
windir = Space(255)  ' make room in the buffer
retval = GetWindowsDirectory(windir, 255)  ' get the path name
windir = Left(windir, InStr(windir, vbNullChar) - 1)  ' trim the null character and unused characters
cloupath = windir & "Clouds.bmp"  ' add the filename to the path
' Set the Windows wallpaper, saving the change and notifying all programs
retval = SystemParametersInfo (SPI_SETDESKWALLPAPER, 0, ByVal cloupath,
SPIF_SENDWININICHANGE Or SPIF_UPDATEINIFILE)

\textbf{Example #2:}

' Tell the user whether the MouseKeys accessibility feature is on
' or off. Note how the structure's cbSize parameter must be set first.
Dim mk As MOUSEKEYS  ' holds settings for MouseKeys
Dim retval As Long  ' return value
mk.cbSize = Len(mk)  ' set the size of the structure
  ' Load the MouseKeys settings into the structure.
retval = SystemParametersInfo(SPI_GETMOUSEKEYS, Len(mk), mk, 0)  ' don't need to notify
  ' Display whether MouseKeys is on or off.
If (mk.dwFlags And MKF_MOUSEKEYSON) = MKF_MOUSEKEYSON Then
  Debug.Print "MouseKeys is on."
Else
  Debug.Print "MouseKeys is off."
End If

See Also: GetSystemMetrics
Category: Accessibility

Go back to the alphabetical Function listing.
Go back to the Reference section index.
SystemTimeToFileTime Function

Declare Function SystemTimeToFileTime Lib "kernel32.dll" (lpSystemTime As SYSTEMTIME, lpFileTime As FILETIME) As Long

Platforms: Win 32s, Win 95/98, Win NT

SystemTimeToFileTime converts a time and date stored in a SYSTEMTIME structure to an identical time and date stored in a FILETIME structure. The former structure provides a easier way to access a date and time, whereas the latter is used by Windows to identify times and dates associated with files. The data put into the FILETIME structure identifies the same time and date as the source structure does. The function returns 0 if an error occurred, or 1 if successful.

lpSystemTime
The date and time, in SYSTEMTIME format, to convert.

lpFileTime
Receives the date and time converted to FILETIME format.

Example:

' Determine if file C:\MyProgram\datafile.txt was created before
' Jan 5, 1999.  Note how CreateFile's alternate declare must be used under Win 95/98 -
' see that function's page for more information.
Dim hfile As Long  ' receives the handle to the file
Dim ctime As FILETIME  ' receives creation date and time of the file
Dim atime As FILETIME  ' receives last access date and time of the file
Dim wtime As FILETIME  ' receives last write-to date and time of the file
Dim jantime As SYSTEMTIME  ' will be set to Jan 5, 1999
Dim janfiletime As FILETIME  ' will receive analogous time as jantime
Dim comptimes As Long  ' receives comparison of ctime and janfiletime
Dim retval As Long  ' return value

' Get a handle to the file (note how the alternate declare is used):
    hfile = CreateFileNS("C:\MyProgram\datafile.txt", GENERIC_READ, FILE_SHARE_READ, 0,
OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hfile = -1 Then ' if the file could not be opened
    Debug.Print "Could not open the file C:\MyProgram\datafile.txt."
End ' abort the program
End If

' Get the various times and dates associated with the file:
    retval = GetFileTime(hfile, ctime, atime, wtime)
' Load jantime with the date January 5, 1999 at midnight:
    jantime.wMonth = 1: jantime.wDay = 5: jantime.wYear = 1999
jantime.wHour = 0: jantime.wMinute = 0: jantime.wSecond = 0
' Convert jantime into FILETIME format so it can be compared with ctime:
retval = SystemTimeToFileTime(jantime, janfiletime)
' Compare the two times and display the relation:
comptimes = CompareFileTime(ctime, janfiletime)
If comptimes = -1 Then Debug.Print "File was created before midnight, January 5, 1999."
If comptimes = 0 Then Debug.Print "File was created at midnight, January 5, 1999."
If comptimes = 1 Then Debug.Print "File was created after midnight, January 5, 1999."
' Close the file
retval = CloseHandle(hfile)

See Also: FileTimeToSystemTime
Category: Time

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/systemtimetofiletime.html
TextOut Function

Declare Function TextOut Lib "gdi32.dll" Alias "TextOutA" (ByVal hdc As Long, ByVal x As Long, ByVal y As Long, ByVal lpString As String, ByVal nCount As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

TextOut displays a line of text on a device. The relation of the text to the (x,y) pair passed to the function can be set using SetTextAlign. The text will be displayed using the device's currently selected font and text drawing color. The function returns 1 if successful, or 0 if an error occurred.

hdc
   The device context of the device to display the line of text on.

x
   The x coordinate of the reference point to display the text at.

y
   The y coordinate of the reference point to display the text at.

lpString
   The string to display on the device.

nCount
   The size in characters of lpString.

Example:

' Display the text "Hello, world!" on window Form1 at (100,50).
' Center the text horizontally at that point and have it appear below the point.
Dim retval As Long  ' return value

' Set the reference point to be centered horizontally and on the top edge of the text:
retval = SetTextAlign(Form1.hDC, TA_CENTER Or TA_TOP Or TA_NOUPDATECP)
' Display the text:
retval = TextOut(Form1.hDC, 100, 50, "Hello, world!", 13)

See Also: GetTextAlign, SetTextAlign
Category: Fonts & Text

Go back to the alphabetical Function listing.
Go back to the Reference section index.
TrackPopupMenu Function

Declare Function TrackPopupMenu Lib "user32.dll" (ByVal hMenu As Long, ByVal uFlags As Long, ByVal x As Long, ByVal y As Long, ByVal nReserved As Long, ByVal hWnd As Long, ByVal prcRect As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

TrackPopupMenu displays a popup menu at a specified point. The function also tracks the menu, updating the selection highlight until the user either selects an item or otherwise closes the menu. By default, the function sends a WM_COMMAND message to the parent window, notifying it of the selection. However, flags specified in the uFlags parameter can change this behavior.

Return Value

If uFlags contains the TPM_RETURNCMD flag, the function returns the identifier of the menu item that the user selected. If an error occurs or the user does not select an item, the function instead returns 0.

If uFlags does not contain the TPM_RETURNCMD flag, the function returns a nonzero value if the function is successful. If an error occurred, the function returns 0. Regardless of uFlags, use the GetLastError to get the error code.

Visual Basic-Specific Issues

None.

Parameters

hMenu
A handle to the popup menu to display and track.

uFlags
A combination of the following flags specifying the positioning of the popup menu and other behavior of the function:

TPM_CENTERALIGN
Position the popup menu so that its horizontal center occurs at the coordinate specified by \( x \).

**TPM_LEFTALIGN**
Position the popup menu so that its left edge occurs at the coordinate specified by \( x \).

**TPM_RIGHTALIGN**
Position the popup menu so that its right edge occurs at the coordinate specified by \( x \).

**TPM_BOTTOMALIGN**
Position the popup menu so that its bottom edge occurs at the coordinate specified by \( y \).

**TPM_TOPALIGN**
Position the popup menu so that its top edge occurs at the coordinate specified by \( y \).

**TPM_VCENTERALIGN**
Position the popup menu so that its vertical center occurs at the coordinate specified by \( y \).

**TPM_NONOTIFY**
Do not send a [WM_COMMAND](http://216.26.168.92/vbapi/ref/t/trackpopupmenu.html) message to the popup menu's parent window to notify it of the user's selection.

**TPM_RETURNCMD**
Have [TrackPopupMenuEx](http://216.26.168.92/vbapi/ref/t/trackpopupmenu.html) return the identifier of the menu item that the user selected.

**TPM_LEFTBUTTON**
Only allow the user to select a menu item using the left mouse button (or the keyboard).

**TPM_RIGHTBUTTON**
Allow the user to select a menu item using either mouse button (or the keyboard).

\( x \)

The x-coordinate of the point where the popup menu is to be displayed, relative to the screen.

\( y \)

The y-coordinate of the point where the popup menu is to be displayed, relative to the screen.

**hWnd**
A handle to the window that owns the popup menu. This window will receive the [WM_COMMAND](http://216.26.168.92/vbapi/ref/t/trackpopupmenu.html) message. The window must be specified even if \( fuFlags \) contains the TPM_NONOTIFY flag.

**prcRect**
Ignored -- set to 0.

## Constant Definitions

```
Const TPM_CENTERALIGN = &H4
Const TPM_LEFTALIGN = &H0
Const TPM_RIGHTALIGN = &H8
Const TPM_BOTTOMALIGN = &H20
Const TPM_TOPALIGN = &H0
Const TPM_VCENTERALIGN = &H10
Const TPM_NONOTIFY = &H80
Const TPM_RETURNCMD = &H100
Const TPM_LEFTBUTTON = &H0
Const TPM_RIGHTBUTTON = &H2
```

## Example

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/t/trackpopupmenu.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreatePopupMenu Lib "user32.dll" () As Long
Public Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long
Windows API Guide: TrackPopupMenu Function

Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type

Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_DEFAULT = &H1000
Public Const MFS_ENABLED = &H0

Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
    MENUITEMINFO) As Long

Public Declare Function TrackPopupMenu Lib "user32.dll" (ByVal hMenu As Long, ByVal uFlags As Long, ByVal x As Long, ByVal y As Long, ByVal nReserved As Long, ByVal hWnd As Long, ByVal prcRect As Long) As Long

Public Const TPM_LEFTALIGN = &H0
Public Const TPM_TOPALIGN = &H0
Public Const TPM_NONOTIFY = &H80
Public Const TPM_RETURNCMD = &H100
Public Const TPM_LEFTBUTTON = &H0

Public Type POINT_TYPE
    x As Long
    y As Long
End Type

Public Declare Function GetCursorPos Lib "user32.dll" (lpPoint As POINT_TYPE) As Long

' When the user clicks button Command1, have a very simple popup menu appear. The menu only has two options, divided by a separator bar. The menu is created when needed and is destroyed after its use.

' The following application-defined constants are used to name the menu item identifiers used by this example. They are not actually part of the API; instead, they are used just to eliminate "magic numbers."
Private Const ID_ABOUT = 101
Private Const ID_SEPARATOR = 102
Private Const ID_EXIT = 103

Private Sub Command1_Click()
Dim hPopupMenu As Long    ' handle to the popup menu to display
Dim mii As MENUITEMINFO  ' describes menu items to add
Dim curpos As POINT_TYPE ' holds the current mouse coordinates
Dim menusel As Long       ' ID of what the user selected in the popup menu
Dim retval As Long        ' generic return value

' Create the popup menu that will be displayed.
hPopupMenu = CreatePopupMenu()
' Add the menu's first item: "About This Problem..."
With mii
    ' The size of this structure.
    .cbSize = Len(mii)
    ' Which elements of the structure to use.
    .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
    ' The type of item: a string.
    .fType = MFT_STRING
    ' This item is currently enabled and is the default item.
    .fState = MFS_ENABLED Or MFS_DEFAULT
    ' Assign this item an item identifier.
    .wID = ID_ABOUT
    ' Display the following text for the item.
    .dwTypeData = "&About This Example..."
    .cch = Len(.dwTypeData)
End With
retval = InsertMenuItem(hPopupMenu, 0, 1, mii)
' Add the second item: a separator bar.
With mii
    .fType = MFT_SEPARATOR
    .fState = MFS_ENABLED
    .wID = ID_SEPARATOR
End With
retval = InsertMenuItem(hPopupMenu, 1, 1, mii)
' Add the final item: "Exit".
With mii
    .fType= MFT_STRING
    .wID = ID_EXIT
    .dwTypeData = "E&xit"
    .cch = Len(.dwTypeData)
End With
retval = InsertMenuItem(hPopupMenu, 2, 1, mii)

' Determine where the mouse cursor currently is, in order to have
' the popup menu appear at that point.
retval = GetCursorPos(curpos)

' Display the popup menu at the mouse cursor. Instead of sending messages
' to window Form1, have the function merely return the ID of the user's
' selection.
menusel = TrackPopupMenu(hPopupMenu, TPM_TOPALIGN Or TPM_LEFTALIGN Or
TPM_NONOTIFY Or TPM_RETURNCMD Or TPM_LEFTBUTTON, curpos.x, curpos.y, 0,
Form1.hWnd, 0)
Before acting upon the user's selection, destroy the popup menu now.

retval = DestroyMenu(hPopupMenu)
Select Case menusel
Case ID_ABOUT
    ' Use the Visual Basic MsgBox function to display a short message in a dialog
    retval = MsgBox("This example demonstrates how to use the API to display " & _
                    "a pop-up menu.", vbOkOnly Or vbInformation, "Windows API Guide")
Case ID_EXIT
    ' End this program by closing and unloading Form1.
    Unload Form1
End Select
End Sub

See Also

TrackPopupMenuEx

Category

Menus

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/t/trackpopupmenu.html
TrackPopupMenuEx Function

Declare Function TrackPopupMenuEx Lib "user32.dll" (ByVal hMenu As Long, ByVal fuFlags As Long, ByVal x As Long, ByVal y As Long, ByVal hwnd As Long, lptpm As TPMPARAMS) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

TrackPopupMenuEx displays a popup menu at a specified point. The function also tracks the menu, updating the selection highlight until the user either selects an item or otherwise closes the menu. By default, the function sends a **WM_COMMAND** message to the parent window, notifying it of the selection. However, flags specified in the *fuFlags* parameter can change this behavior.

Return Value

If *fuFlags* contains the TPM_RETURNCMD flag, the function returns the identifier of the menu item that the user selected. If an error occurs or the user does not select an item, the function instead returns 0.

If *fuFlags* does not contain the TPM_RETURNCMD flag, the function returns a nonzero value if the function is successful. If an error occurred, the function returns 0. Regardless of *fuFlags*, use the **GetLastError** to get the error code.

Visual Basic-Specific Issues

None.

Parameters

*hMenu*

A handle to the popup menu to display and track.

*fuFlags*

A combination of the following flags specifying the positioning of the popup menu and other behavior of the function: TPM_CENTERALIGN
Position the popup menu so that its horizontal center occurs at the coordinate specified by $x$.

**TPM_LEFTALIGN**
Position the popup menu so that its left edge occurs at the coordinate specified by $x$.

**TPM_RIGHTALIGN**
Position the popup menu so that its right edge occurs at the coordinate specified by $x$.

**TPM_BOTTOMALIGN**
Position the popup menu so that its bottom edge occurs at the coordinate specified by $y$.

**TPM_TOPALIGN**
Position the popup menu so that its top edge occurs at the coordinate specified by $y$.

**TPM_VCENTERALIGN**
Position the popup menu so that its vertical center occurs at the coordinate specified by $y$.

**TPM_NONOTIFY**
Do not send a **WM_COMMAND** message to the popup menu's parent window to notify it of the user's selection.

**TPM_RETURNCMD**
Have **TrackPopupMenuEx** return the identifier of the menu item that the user selected.

**TPM_LEFTBUTTON**
Only allow the user to select a menu item using the left mouse button (or the keyboard).

**TPM_RIGHTBUTTON**
Allow the user to select a menu item using either mouse button (or the keyboard).

**TPM_HORIZONTAL**
If the popup menu cannot be shown at the desired coordinates without intruding upon the exclusion rectangle, preserve the horizontal positioning instead of the vertical.

**TPM_VERTICAL**
If the popup menu cannot be shown at the desired coordinates without intruding upon the exclusion rectangle, preserve the vertical positioning instead of the horizontal.

$x$
The $x$-coordinate of the point where the popup menu is to be displayed, relative to the screen.

$y$
The $y$-coordinate of the point where the popup menu is to be displayed, relative to the screen.

$hwnd$
A handle to the window that owns the popup menu. This window will receive the **WM_COMMAND** message. The window must be specified even if $fuFlags$ contains the TPM_NONOTIFY flag.

$lptpm$
Specified additional information about the positioning of the popup menu, specifically regarding the exclusion rectangle. The exclusion rectangle is a region of the screen in which the popup menu will not appear.

## Constant Definitions

```plaintext
Const TPM_CENTERALIGN = &H4
Const TPM_LEFTALIGN = &H0
Const TPM_RIGHTALIGN = &H8
Const TPM_BOTTOMALIGN = &H20
Const TPM_TOPALIGN = &H0
Const TPM_VCENTERALIGN = &H10
Const TPM_NONOTIFY = &H80
Const TPM_RETURNCMD = &H100
Const TPM_LEFTBUTTON = &H0
Const TPM_RIGHTBUTTON = &H2
Const TPM_HORIZONTAL = &H0
Const TPM_VERTICAL = &H40
```
Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function CreatePopupMenu Lib "user32.dll" () As Long
Public Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_DEFAULT = &H1000
Public Const MFS_ENABLED = &H0
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As _
    MENUITEMINFO) As Long
Public Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type
Public Type TPMPARAMS
    cbSize As Long
    rcExclude As RECT
End Type
Public Declare Function TrackPopupMenuEx Lib "user32.dll" (ByVal hMenu As Long, ByVal _
    fuFlags As Long, ByVal x As Long, ByVal y As Long, ByVal hWnd As Long, lptpm As _
    TPMPARAMS) As Long
Public Const TPM_LEFTALIGN = &H0
Public Const TPM_TOPALIGN = &H0
Public Const TPM_NONOTIFY = &H80
Public Const TPM_RETURNCMD = &H100
Public Const TPM_LEFTBUTTON = &H0
Public Type POINT_TYPE
    x As Long
    y As Long
End Type
Public Declare Function GetCursorPos Lib "user32.dll" (lpPoint As POINT_TYPE) As Long
Public Declare Function SetRectEmpty Lib "user32.dll" (lpRect As RECT) As Long

' When the user clicks button Command1, have a very simple popup menu appear. The menu only has two options, divided by a separator bar. The menu is created when needed and is destroyed after its use.

' The following application-defined constants are used to name the menu item identifiers used by this example. They are not actually part of the API; instead, they are used just to eliminate "magic numbers."
Private Const ID_ABOUT = 101
Private Const ID_SEPARATOR = 102
Private Const ID_EXIT = 103

Private Sub Command1_Click()
    Dim hPopupMenu As Long    ' handle to the popup menu to display
    Dim mii As MENUITEMINFO   ' describes menu items to add
    Dim tpm As TPMPARAMS      ' identifies the exclusion rectangle
    Dim curpos As POINT_TYPE  ' holds the current mouse coordinates
    Dim menusel As Long       ' ID of what the user selected in the popup menu
    Dim retval As Long        ' generic return value

    ' Create the popup menu that will be displayed.
    hPopupMenu = CreatePopupMenu()
    ' Add the menu's first item: "About This Problem..."
    With mii
        ' The size of this structure.
        .cbSize = Len(mii)
        ' Which elements of the structure to use.
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        ' The type of item: a string.
        .fType = MFT_STRING
        ' This item is currently enabled and is the default item.
        .fState = MFS_ENABLED Or MFS_DEFAULT
        ' Assign this item an item identifier.
        .wID = ID_ABOUT
        ' Display the following text for the item.
        .dwTypeData = "&About This Example..."
        .cch = Len(.dwTypeData)
    End With
    retval = InsertMenuItem(hPopupMenu, 0, 1, mii)
    ' Add the second item: a separator bar.
    With mii
        .fType = MFT_SEPARATOR
        .fState = MFS_ENABLED
    End With

.wID = ID_SEPARATOR
End With
retval = InsertMenu(hPopupMenu, 1, 1, mii)
' Add the final item: "Exit".
With mii
  .fType= MFT_STRING
  .wID = ID_EXIT
  .dwTypeData = "E&xit"
  .cch = Len(.dwTypeData)
End With
retval = InsertMenu(hPopupMenu, 2, 1, mii)

' Determine where the mouse cursor currently is, in order to have
' the popup menu appear at that point.
retval = GetCursorPos(curpos)

' Make the exclusion rectangle empty because there's no need for it here.
With tpm
  ' Size of the structure.
  .cbSize = Len(tpm)
  ' Make the exclusion rectangle empty.
  retval = SetRectEmpty(.rcExclude)
End With

' Display the popup menu at the mouse cursor. Instead of sending messages
' to window Form1, have the function merely return the ID of the user's
' selection.
menusel = TrackPopupMenuEx(hPopupMenu, TPM_TOPALIGN Or TPM_LEFTALIGN Or
  TPM_NONOTIFY _
  Or TPM_RETURNCMD Or TPM_LEFTBUTTON, curpos.x, curpos.y, Form1.hWnd, tpm)

' Before acting upon the user's selection, destroy the popup menu now.
retval = DestroyMenu(hPopupMenu)
Select Case menusel
Case ID_ABOUT
  ' Use the Visual Basic MsgBox function to display a short message in
  ' a dialog.
  ' box. Using the MessageBox API function isn't necessary.
  retval = MsgBox("This example demonstrates how to use the API to
display " & _
                "a pop-up menu.", vbOkOnly Or vbInformation, "Windows API
Guide")
Case ID_EXIT
  ' End this program by closing and unloading Form1.
  Unload Form1
End Select
End Sub

See Also
TrackPopupMenu

Category

Menus

Back to the Function list.
Back to the Reference section.
UnionRect Function

Declare Function UnionRect Lib "user32.dll" (lpDestRect As RECT, lpSrc1Rect As RECT, lpSrc2Rect As RECT) As Long

Platforms: Win 32s, Win 95/98, Win NT

UnionRect determines the smallest possible rectangle that contains two other rectangles. This rectangle is called the union rectangle because it is derived from the union of the areas that the two source rectangles occupy. The union rectangle is put into the variable passed as lpDestRect. The function returns 0 if an error occurred, or 1 if successful.

lpDestRect
Variable that receives the union rectangle.

lpSrc1Rect
The first of the two source rectangles.

lpSrc2Rect
The second of the two source rectangles.

Example:

' Create a large rectangle that contains the areas occupied by rectangles s and t. The new rectangle will fully contain both smaller rectangles.
' s = (20,30)-(60,80); t = (100,110)-(200,300)
Dim s As RECT, t As RECT  ' small rectangles
Dim urect As RECT  ' receives the union rectangle
Dim retval As Long  ' return value

' Set the small rectangles
retval = SetRect(s, 20, 30, 60, 80)  ' set s = (20,30)-(60,80)
retval = SetRect(t, 100, 110, 200, 300)  ' set t = (100,110)-(200,300)

' Figure out the union rectangle
retval = UnionRect(urect, s, t)  ' now urect = (20,30)-(200,300)

See Also: IntersectRect, SubtractRect

Category: Rectangles

Go back to the alphabetical Function listing.
Go back to the Reference section index.
UnregisterClass Function

Declare Function UnregisterClass Lib "user32.dll" Alias "UnregisterClassA" (ByVal lpClassName As Any, ByVal hInstance As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

UnregisterClass unregisters a window class, after which it can no longer be used. Obviously, a window class cannot be unregistered if windows still exist which belong to the class. All window classes created by an application should be unregistered after the application is completely finished using them.

Return Value

If an error occurred, the function returns 0 (use GetLastError to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

*lpClassName*

The name of the window class to unregister, or an atom identifying the window class to unregister.

*hInstance*

A handle to the instance which originally registered the window class.

Example

' This code is licensed according to the terms and conditions listed here.

' Register a new window class. The various properties to
give it will be explained in the code.

' *** Place the following code in a module. ***
' Define the window procedure to use for the class. Here, we'll
' just make a wrapper for the default window procedure.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As
Long, ByVal lParam As Long) As Long
' Return whatever the default window procedure returns.
WindowProc = DefWindowProc(hWnd, uMsg, wParam, lParam)
End Function

' *** Place the following code where you want to register the class. ***
Dim classinfo As WNDCLASSEX ' holds info about the class
Dim classatom As Long ' receives an atom to the class just registered

' Set the various data members of the structure.
' Of course, set the size of the structure.
classinfo.cbSize = Len(classinfo)
' Class style: give each window its own DC; redraw when resized.
classinfo.style = CS_OWNDC Or CS_HREDRAW Or CS_VREDRAW
' Use the window procedure above to process messages.
classinfo.lpfnWndProc = AddressOf WindowProc
' We aren't using any extra information.
classinfo.cbClsExtra = 0
classinfo.cbWndExtra = 0
' Handle to the instance of this application.
classinfo.hInstance = App.hInstance
' Use the icon stored in C:\MyApp\deficon.ico.
classinfo.hIcon = ExtractIcon(App.hInstance, "C:\MyApp\deficon.ico", 0)
' Use the cursor stored in C:\MyApp\mouse.cur.
classinfo.hCursor = LoadCursorFromFile("C:\MyApp\mouse.cur")
' Fill the background with the system color for an application's workspace.
classinfo.hbrBackground = COLOR_APPWORKSPACE
' No menu resource to use.
classinfo.lpszMenuName = 
' Give the class a name.
classinfo.lpszClassName = "CustomClass"
' Use the small icon stored in C:\MyApp\deficonsm.ico.
classinfo.hIconSm = ExtractIcon(App.hInstance, "C:\MyApp\deficonsm.ico", 0)

' Finally, register the class.
classatom = RegisterClassEx(classinfo)
' Now the class CustomClass can be used to create windows.

' *** Place the following code where you wish to unregister the window class. ***
Dim retval As Long

Unregister the window class.
retval = UnregisterClass("CustomClass", App.hInstance)
See Also

RegisterClass, RegisterClassEx

Category

Window Classes

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 24, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/u/unregisterclass.html
VerQueryValue Function

Declare Function VerQueryValue Lib "version.dll" Alias "VerQueryValueA" (pBlock As Any, ByVal lpSubBlock As String, lplpBuffer As Long, puLen As Long) As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

VerQueryValue extracts information from a version information resource. This resource stores version-related information about a 32-bit executable-type file. The GetFileVersionInfo function obtains this resource from the desired file. VerQueryValue can get both general version information and version identifying strings from the resource. Naturally, since version information resources do not exist for non-32-bit applications, nor for non-executable files, this function will not work with them.

Return Value

If successful, the function returns a non-zero value. If an error occurred, the function returns 0.

Visual Basic-Specific Issues

To actually use the information retrieved by VerQueryValue, you have to copy the data identified by the pointer lplpBuffer into the appropriate object. To do this, use lstrcpy for strings and CopyMemory for anything else. For the root block, copy the pointer's data into a VS_FIXEDFILEINFO structure. For the code page and language data, copy the data into a four-element Byte array. For version information strings, copy the data into, you guessed it, a string.

If you're a bit confused or overwhelmed, don't be. The example on this page illustrates how to do all three.

Parameters

pBlock

The byte array or similar object that holds the version information resource extracted from a file. To obtain such a block, use GetFileVersionInfo.

lpSubBlock

One of the following strings specifying what information to extract from the resource. A pointer to the extracted information is placed into `lplpBuffer`.

"\"

Extract the root block of version information, which is in the form of a `VS_FIXEDFILEINFO` structure.

"\VarFileInfo\Translation"

Extract the identifier of the language and code page in which the version information strings are encoded. While this is not useful in and of itself, this data is necessary to read any version information strings in the resource. See Note 1 below for details about how to use this value.

"\StringFileInfo\lang-codepage\string-name"

Extract one of the version information strings in the resource. `lang-codepage` is an 8-digit hexadecimal value that identifies the code page and language used to encode the string. `string-name` is one of the following values that identify the various strings available:

- "Comments"
- "CompanyName"
- "FileDescription"
- "FileVersion"
- "InternalName"
- "LegalCopyright"
- "LegalTrademarks"
- "OriginalFilename"
- "PrivateBuild"
- "ProductName"
- "ProductVersion"
- "SpecialBuild"

`lplpBuffer`

Receives a pointer to the data extracted from the version information resource. The memory referenced by this pointer is automatically freed when the memory used by `pBlock` is freed. In other words, don't worry about freeing this memory after you use it.

`puLen`

Receives the size in bytes of the data referenced by the `lplpBuffer` pointer.

Note 1

The code page and language data comes as a series of four bytes. Before using the data, however, it needs to be modified slightly. You must swap the first two bytes with each other, and then swap the last two with each other. Then, you need to convert the four bytes into an 8-digit hexadecimal string, with the first byte (after the swap) at the beginning and the last byte at the end. For example, if "00" represents the first byte, "11" the second, etc., the string will look like "00112233". You are now ready to use this string as the `lang-codepage` component of a version information string identifier. Go back up.

Example

Display information about the file whose full path and filename is entered into textbox Text1. Display the version number, copyright information, and file description when button Command1 is pressed. To use this example, you obviously have to enter the filename of a 32-bit executable/DLL/etc. into Text1. Obviously, this example requires that you create a text box called Text1 and a command button called Command1.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
(ByVal lptstrFilename As String, ByVal dwHandle As Long, ByVal dwLen As Long, _
lpData As Any) As Long
Public Declare Function GetFileVersionInfoSize Lib "version.dll" Alias "GetFileVersionInfoSizeA" (ByVal lptstrFilename As String, lpdwHandle As Long) As Long
Public Declare Function VerQueryValue Lib "version.dll" Alias "VerQueryValueA" (pBlock _
As Any, ByVal lpSubBlock As String, lplpBuffer As Long, puLen As Long) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 _
As Any, ByVal lpString2 As Any) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination _
As Any, ByVal lpString2 As Any) As Long
Public Type VS_FIXEDFILEINFO
  dwSignature As Long
  dwStrucVersion As Long
  dwFileVersionMS As Long
  dwFileVersionLS As Long
  dwProductVersionMS As Long
  dwProductVersionLS As Long
  dwFileFlagsMask As Long
  dwFileFlags As Long
  dwFileOS As Long
  dwFileType As Long
  dwFileSubtype As Long
  dwFileDateMS As Long
  dwFileDateLS As Long
End Type
Public Const VFT_APP = &H1
Public Const VFT_DLL = &H2
Public Const VFT_DRV = &H3
Public Const VFT_VXD = &H5

' *** Place the following function definitions inside a module. ***

' HIWORD and LOWORD are API macros defined below.
Public Function HIWORD (ByVal dwValue As Long) As Long
  Dim hexstr As String
  hexstr = Right("00000000" & Hex(dwValue), 8)
  HIWORD = CLng("&H" & Left(hexstr, 4))
End Function
Public Function LOWORD (ByVal dwValue As Long) As Long
  Dim hexstr As String
  hexstr = Right("00000000" & Hex(dwValue), 8)
  LOWORD = CLng("&H" & Right(hexstr, 4))
End Function
'This nifty subroutine swaps two byte values without needing a buffer variable. This technique, which uses Xor, works as long as the two values to be swapped are numeric and of the same data type (here, both Byte).

Public Sub SwapByte (byte1 As Byte, byte2 As Byte)
    byte1 = byte1 Xor byte2
    byte2 = byte1 Xor byte2
    byte1 = byte1 Xor byte2
End Sub

This function creates a hexadecimal string to represent a number, but it outputs a string of a fixed number of digits. Extra zeros are added to make the string the proper length. The "&H" prefix is not put into the string.

Public Function FixedHex (ByVal hexval As Long, ByVal nDigits As Long) As String
    FixedHex = Right("00000000" & Hex(hexval), nDigits)
End Function

*** Place the following code inside the form that has Command1 and Text1. ***

Private Sub Command1_Click()
    Dim vffi As VS_FIXEDFILEINFO ' version info structure
    Dim buffer() As Byte ' buffer for version info resource
    Dim pData As Long ' pointer to version info data
    Dim nDataLen As Long ' length of info pointed at by pData
    Dim cpl(0 To 3) As Byte ' buffer for code page & language
    Dim cplstr As String ' 8-digit hex string of cpl
    Dim dispstr As String ' string used to display version information
    Dim retval As Long ' generic return value

    ' First, get the size of the version info resource. If this function fails, then Text1 identifies a file that isn't a 32-bit executable/DLL/etc.
    nDataLen = GetFileVersionInfoSize(Text1.Text, pData)
    If nDataLen = 0 Then
        Debug.Print "Not a 32-bit executable!"
        Exit Sub
    End If

    ' Make the buffer large enough to hold the version info resource.
    ReDim buffer(0 To nDataLen - 1) As Byte

    ' Get the version information resource.
    retval = GetFileVersionInfo(Text1.Text, 0, nDataLen, buffer(0))

    ' Get a pointer to a structure that holds a bunch of data.
    retval = VerQueryValue(buffer(0), ",", pData, nDataLen)

    ' Copy that structure into the one we can access.
    CopyMemory vffi, ByVal pData, nDataLen

    ' Display the full version number of the file.
    dispstr = Trim(Str(HIWORD(vffi.dwFileVersionMS))) & "." & _
        Trim(Str(LOWORD(vffi.dwFileVersionMS))) & "." & _
        Trim(Str(HIWORD(vffi.dwFileVersionLS))) & "." & _
        Trim(Str(LOWORD(vffi.dwFileVersionLS)))

    Debug.Print "Version Number: "; dispstr ' Display the type of file it is (i.e., executable, DLL, etc.).
Select Case vffi.dwFileType
Case VFT_APP
    dispstr = "Application"
Case VFT_DLL
    dispstr = "Dynamic Link Library (DLL)"
Case VFT_DRV
    dispstr = "Device Driver"
Case VFT_VXD
    dispstr = "Virtual Device Driver"
Case Else
    dispstr = "Unknown"
End Select
Debug.Print "File Type: "; dispstr

' Before reading any strings out of the resource, we must first determine the code page
' and language. The code to get this information follows.
retval = VerQueryValue(buffer(0), "\VarFileInfo\Translation", pData, nDataLen)
' Copy that information into the byte array.
CopyMemory cpl(0), ByVal pData, 4
' It is necessary to swap the first two bytes, as well as the last two bytes.
SwapByte cpl(0), cpl(1)
SwapByte cpl(2), cpl(3)
' Convert those four bytes into a 8-digit hexadecimal string.
cplstr = FixedHex(cpl(0), 2) & FixedHex(cpl(1), 2) & FixedHex(cpl(2), 2) & _
    FixedHex(cpl(3), 2)
' cplstr now represents the code page and language to read strings as.
'
' Read the copyright information from the version info resource.
retval = VerQueryValue(buffer(0), "\StringFileInfo\" & cplstr & "\LegalCopyright", _
    pData, nDataLen)
' Copy that data into a string for display.
dispstr = Space(nDataLen)
retval = lstrcpy(dispstr, pData)
' Display the result.
Debug.Print "Copyright Info: "; dispstr
' Similarly, read a description of the file and display it.
retval = VerQueryValue(buffer(0), "\StringFileInfo\" & cplstr & "\FileDescription", _
    pData, nDataLen)
dispstr = Space(nDataLen)
retval = lstrcpy(dispstr, pData)
Debug.Print "File Description: "; dispstr
End Sub

See Also
GetFileVersionInfo, GetFileVersionInfoSize
WaitForSingleObject Function

Declare Function WaitForSingleObject Lib "kernel32.dll" (ByVal hHandle As Long, ByVal dwMilliseconds As Long) As Long

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

`WaitForSingleObject` halts program execution temporarily, until either a specified object enters a signaled state or a specified timeout period elapses. This function allows a program to wait until something else has happened before execution continues.

Return Value

The function returns one of the following flags:

- **WAIT_ABANDONED**
  - The mutex specified by `hHandle` was not released by its owning thread before that thread terminated. The mutex is now owned by the calling thread and is in a nonsignaled state.

- **WAIT_FAILED**
  - The function failed. Use `GetLastError` to get the error code.

- **WAIT_OBJECT_0**
  - The object specified by `hHandle` is in a signaled state.

- **WAIT_TIMEOUT**
  - The timeout period has elapsed.

Visual Basic-Specific Issues

None.

Parameters
**hHandle**

A handle to the object to wait for to enter into a signaled state. This handle can refer to one of the following types of objects:

- Change Notification
- Console Input
- Event
- Job (is signaled when the job has finished)
- Mutex
- Process (is signaled when the process has terminated)
- Semaphore
- Thread (is signaled when the thread has terminated)
- Waitable timer

**dwMilliseconds**

The timeout interval to wait for. The function will return (with a value of WAIT_TIMEOUT) if this timeout period elapses, even if the object being waited for has not entered a single state. This parameter could also be the following value:

- Infinite
  
  Wait forever -- do not use a timeout period.

---

## Constant Definitions

```vbscript
Const WAIT_ABANDONED = &80
Const WAIT_FAILED = &HFFFFFFFF
Const WAIT_OBJECT_0 = &H0
Const WAIT_TIMEOUT = &H102
Const INFINITE = &HFFFF
```

---

## Example

Open the file "C:\Docs\readme.txt" using whatever program is associated with *.txt files (by default, Notepad). Wait until the user has closed the window before continuing with the example. The example begins when the user clicks button Command1. Obviously, to use this example, you need to place a command button named Command1 on a form window.

```
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SHELLEXECUTEINFO
    cbSize As Long
    fMask As Long
    hwnd As Long
    lpVerb As String
    lpFile As String
    lpParameters As String
    lpDirectory As String
    nShow As Long
    hInstApp As Long
    lpIDList As Long
    lpClass As String
End Type
```
hkeyClass As Long
dwHotKey As Long
hIcon As Long
hProcess As Long

End Type
Public Const SEE_MASK_NOCLOSEPROCESS = &H40
Public Const SW_SHOWNORMAL = 1
Public Declare Function ShellExecuteEx Lib "shell32.dll" Alias "ShellExecuteExA"
(lpExecInfo As _
    SHELLEXECUTEINFO) As Long
Public Const SE_ERR_FNF = 2
Public Const SE_ERR_NOASSOC = 31
Public Declare Function WaitForSingleObject Lib "kernel32.dll" (ByVal hHandle As
    Long, ByVal dwMilliseconds As Long) As Long
Public Const INFINITE = &HFFFF
Public Const WAIT_TIMEOUT = &H102

' *** Place the following code inside window Form1. ***
Private Sub Command1_Click()
    Dim sei As SHELLEXECUTEINFO ' structure used by the function
    Dim retval As Long ' return value

    ' Load the information needed to open C:\Docs\readme.txt
    ' into the structure.
    With sei
        ' Size of the structure
        .cbSize = Len(sei)
        ' Use the optional hProcess element of the structure.
        .fMask = SEE_MASK_NOCLOSEPROCESS
        ' Handle to the window calling this function.
        .hwnd = Form1.hWnd
        ' The action to perform: open the file.
        .lpVerb = "open"
        ' The file to open.
        .lpFile = "C:\Docs\readme.txt"
        ' No additional parameters are needed here.
        .lpParameters = ""
        ' The default directory -- not really necessary in this case.
        .lpDirectory = "C:\Docs\"
        ' Simply display the window.
        .nShow = SW_SHOWNORMAL
        ' The other elements of the structure are either not used
        ' or will be set when the function returns.
    End With

    ' Open the file using its associated program.
    retval = ShellExecuteEx(sei)
    If retval = 0 Then
        ' The function failed, so report the error. Err.LastDllError
        ' could also be used instead, if you wish.
Select Case sei.hInstApp
Case SE_ERR_FNF
    Debug.Print "The file C:\Docs\readme.txt was not found."
Case SE_NOASSOC
    Debug.Print "No program is associated with *.txt files."
Case Else
    Debug.Print "An unexpected error occurred."
End Select
Else
    ' Wait for the opened process to close before continuing. Instead of waiting once for a time of INFINITE, this example repeatedly checks to see if the
    ' is still open. This allows the DoEvents VB function to be called, preventing
    ' our program from appearing to lock up while it waits.
    Do
        DoEvents
        retval = WaitForSingleObject(sei.hProcess, 0)
    Loop While retval = WAIT_TIMEOUT
    Debug.Print "Notepad (or whatever program was opened) has just closed."
End If
End Sub

Category

Synchronization

Back to the Function list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/waitforsingleobject.html
waveOutGetDevCaps Function

Declare Function waveOutGetDevCaps Lib "winmm.dll" Alias "waveOutGetDevCapsA" (ByVal uDeviceID As Long, lpCaps As WAVEOUTCAPS, ByVal uSize As Long) As Long

Platforms: Win 95/98, Win NT

waveOutGetDevCaps reads the capabilities and other information about a given waveform output device. This information is placed in the structure passed as lpCaps. This function can determine what operations a waveform output device can perform. The function returns 0 if successful, or a non-zero error code if an error occurred.

uDeviceID
The device identifier of the waveform output device to get information about. Remember that the first device has an identifier of 0.

lpCaps
Receives the information about the waveform output device.

uSize
The size in bytes of lpCaps.

' List the names and number of channels of every installed waveform output device. Note how we only check the valid device identifiers.
Dim outinfo As WAVEOUTCAPS ' receives info about each device
Dim numdevs As Long ' number of installed devices
Dim thisdev As Long ' counter for which device we're checking
Dim outname As String ' buffer for device's name
Dim retval As Long ' return value

' First, determine the number of waveform output devices.
numdevs = waveOutGetNumDevs()

' Loop through each device and display the desired information. Keep in mind that the first
device has an identifier of 0, etc.
For thisdev = 0 To numdevs - 1
    ' Get the capabilities of the device
    retval = waveOutGetDevCaps(thisdev, outinfo, Len(outinfo))
    If retval = 0 Then ' only continue if the above function succeeded
        Debug.Print "Device "; thisdev
        ' Extract device name from fixed-length string
        outname = Left(outinfo.szPname, InStr(outinfo.szPname, vbNullChar) - 1)
        Debug.Print " "; outname;
        ' Display number of channels -- i.e., is it mono or stereo?
        If outinfo.nChannels = 1 Then
            Debug.Print " (mono)"
    End If
Else
    Debug.Print " (stereo)"
End If
End If
Next thisdev
waveOutGetNumDevs Function

Declare Function waveOutGetNumDevs Lib "winmm.dll" () As Long

Platforms: Win 95/98, Win NT

waveGetNumDevs find the number of waveform output devices installed on the computer. This function can be used to determine the valid range in device identifiers for waveform output devices. The function returns the number of installed waveform output devices.

Example:

' List the names and number of channels of every installed waveform output device. Note how we only check the valid device identifiers.
Dim outinfo As WAVEOUTCAPS ' receives info about each device
Dim numdevs As Long ' number of installed devices
Dim thisdev As Long ' counter for which device we're checking
Dim outname As String ' buffer for device's name
Dim retval As Long ' return value

' First, determine the number of waveform output devices.
umdevs = waveOutGetNumDevs()
' Loop through each device and display the desired information. Keep in mind that the first device has an identifier of 0, etc.
For thisdev = 0 To numdevs - 1
' Get the capabilities of the device
retval = waveOutGetDevCaps(thisdev, outinfo, Len(outinfo))
If retval = 0 Then ' only continue if the above function succeeded
    Debug.Print "Device "; thisdev
    ' Extract device name from fixed-length string
    outname = Left(outinfo.szPname, InStr(outinfo.szPname, vbNullChar) - 1)
    Debug.Print " "; outname;
    ' Display number of channels -- i.e., is it mono or stereo?
    If outinfo.nChannels = 1 Then
        Debug.Print " (mono)"
    Else
        Debug.Print " (stereo)"
    End If
End If
Next thisdev

Category: Audio
waveOutPutGetVolume Function

Declare Function waveOutPutGetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, lpdwVolume As Long) As Long

Platforms: Win 95/98, Win NT

waveOutPutGetVolume finds the current volume setting for a waveform output device. The volume setting is placed in the variable passed as lpdwVolume and is split into a high-order word and a low-order word. If the device supports separate left and right channel volumes, the low-order word contains the left volume and the high-order word contains the right volume. If it does not, the low-order word contains the overall volume and the high-order word is ignored. Volume values range from silence (&H0) to maximum (&HFFFF). The function returns 0 if successful, or a non-zero error code if an error occured.

uDeviceID
Either the device ID or a handle to the waveform output device to poll the volume of.

lpdwVolume
Receives the volume setting of the device as described above.

Example:

' Display the current volume setting for waveform output device 0. Note
' that we must first determine if separate volumes are returned or not, in order to
' know
' how to interpret the volume returned. (We assume waveform output device #0
' exists.)
Dim volume As Long  ' receives volume(s)
Dim lvolume As Long, rvolume As Long  ' separate channel volumes
Dim spkrcaps As WAVEOUTCAPS  ' needed to find volume interpretation
Dim numvols As Integer  ' will be 1 if only one volume setting or 2 if there are two
Dim retval As Long  ' return value

' First, find out whether the left and right channels have separate volumes.
retval = waveOutPutGetDevCaps(0, spkrcaps, Len(spkrcaps))  ' get information
If (spkrcaps.dwSupport And WAVECAPS_LRVOLUME) = WAVECAPS_LRVOLUME Then
    numvols = 2  ' separate channel volumes
Else
    numvols = 1  ' only one volume
End If

' Get the volume setting(s) and display them in hexadecimal.
retval = waveOutPutGetVolume(0, volume)
If numvols = 1 Then  ' if only one channel volume
    volume = volume And &HFFFF  ' destroy irrelevant high-order word
Debug.Print "Waveform Output Device #0 volume: "; Hex(volume)
Else ' if separate channel volumes
    lvolume = volume And &HFFFF  ' isolate left speaker volume
    rvolume = (volume And &HFFFF0000) / &H10000  ' isolate right speaker volume
Debug.Print "Waveform Output Device #0 left channel volume: "; Hex(lvolume)
Debug.Print "Waveform Output Device #0 right channel volume: "; Hex(rvolume)
End If

See Also: waveOutSetVolume
Category: Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/waveoutgetvolume.html
waveOutSetVolume Function

Declare Function waveOutSetVolume Lib "winmm.dll" (ByVal uDeviceID As Long, ByVal dwVolume As Long) As Long

Platforms: Win 95/98, Win NT

waveOutSetVolume sets the volume level for a waveform output device. Depending on the capabilities of the device, this function either sets the independent left and right channel volumes or the overall volume. Each volume setting must be in the range between &H0 (silence) and &HFFFF (maximum volume). The function returns 0 if successful, or a non-zero error code if and error occured.

uDeviceID
- Either the device identifier or a handle to the waveform output device to set the volume of.

dwVolume
- The new volume setting(s) for the device. If separate left/right volumes are supported, the low-order word is the left channel volume and the high-order word is the right channel volume. If not, the low-order word contains the overall volume and the high-order word is ignored.

Example:

' Set the volume on waveform output device #0 to 50% of maximum.
' Note how we don't care if there are separate volumes or not -- we set both to the
' setting and, if only one channel volume exists, the extra data is ignored.
Dim retval As Long ' return value

' First, verify that waveform output device #0 does exist.
retval = waveOutGetNumDevs() ' get number of such devices
If retval >= 1 Then ' at least one exists, so device #0 is there!
   ' Set the volume for all channels to &H7FFF (50%)
   retval = waveOutSetVolume(0, &H7FFF7FFF) ' for both channels, if needed
End If

See Also: waveOutGetVolume

Category: Audio

Go back to the alphabetical Function listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
WindowFromPoint Function

Declare Function WindowFromPoint Lib "user32.dll" (ByVal xPoint As Long, ByVal yPoint As Long) As Long

Platforms: Win 32s, Win 95/98, Win NT

WindowFromPoint determines the handle of the window located at a specific point on the screen. Note that the active window could be a text box, list box, button, or some other object sitting inside a program window. In this case, the handle returned will be to this control and not the program window. If successful, the function returns the handle to the window at that point. If there is no window at that point, or if an error occurred, the function instead returns 0.

xPoint
The x-coordinate of the point to look for a window at.
yPoint
The y-coordinate of the point to look for a window at.

Example:

' Display the title bar text of whatever window the mouse cursor is currently over. Note that this could be a control on a program window.
Dim mousepos As POINT_TYPE ' coordinates of the mouse cursor
Dim wintext As String, slength As Long ' receive title bar text and its length
Dim hwnd As Long ' handle to the window found at the point
Dim retval As Long ' return value

' Determine the window the mouse cursor is over.
retval = GetCursorPos(mousepos) ' get the location of the mouse
hwnd = WindowFromPoint(mousepos.x, mousepos.y) ' determine the window that's there
If hwnd = 0 Then ' error or no window at that point
    Debug.Print "No window exists at that location."
Else
    slength = GetWindowTextLength(hwnd) ' get length of title bar text
    wintext = Space(slength + 1) ' make room in the buffer to receive the string
    slength = GetWindowText(hwnd, wintext, slength + 1) ' get the text
    wintext = Left(wintext, slength) ' extract the returned string from the buffer
    Debug.Print "Title bar text of the window: "; wintext
End If

Category: Windows
WinHelp Function

Declare Function WinHelp Lib "user32.dll" Alias "WinHelpA" (ByVal hWndMain As Long, ByVal lpHelpFile As String, ByVal uCommand As Long, dwData As Any) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

WinHelp opens a Windows Help file or somehow manipulates an open Help file. The precise action taken by the function depends on the value passed as `uCommand`, but all of them work with Windows Help files. This function is used to provide all access points between an application and its help file(s).

Return Value

If an error occurred, the function returns 0 (use `GetLastError` to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

Whenever passing a String or a Long (not including an array of Longs) as `dwData`, you must precede it with the `ByVal` keyword. Do not use the keyword when passing any data structure or an array.

Parameters

`hWndMain`

In most cases, a `handle` to the window which is opening the Help file. If `uCommand` is either HELP_CONTEXTMENU or HELP_WM_HELP, this is a handle to the particular control to open up.
context-sensitive Help about.

`lpszHelp`

The filename of the Help file to display. The filename can be followed by the > character and the name of a secondary Help window (defined in the Help file) to open instead of the primary one.

`uCommand`

Exactly one of the following flags specifying which action the function is to take on the Help file:

- **HELP_COMMAND**
  - Execute a Help macro or macro string. `dwData` is a string specifying the name(s) of the Help macro(s) to run. If more than one is specified, separate each name with semicolons.

- **HELP_CONTENTS**
  - Display the Contents topic of the Help file. `dwData` must be 0. This flag is obsolete; use the HELP_FINDER flag instead.

- **HELP_CONTEXT**
  - Display the topic identified by the value passed as `dwData`.

- **HELP_CONTEXTMENU**
  - Display the Help topic associated with the window's selected control in a pop-up window. `dwData` is an array of pairs of Longs (dwords): the first in a pair is a control identifier, and the second in a pair is the context identifier of the associated Help topic. The array's last pair must be two zeros.

- **HELP_CONTEXTPOPUP**
  - Display the topic identified by the value passed as `dwData` in a pop-up window.

- **HELP_FINDER**
  - Display the Help Topics dialog box. `dwData` must be 0.

- **HELP_FORCEFILE**
  - Ensure that Windows Help is displaying the correct Help file; if it is not, then display the correct one instead. `dwData` must be 0.

- **HELP_HELPPONHELP**
  - Display the Help on using Windows Help Help file, which is part of Windows. `dwData` must be 0.

- **HELP_INDEX**
  - Same as HELP_CONTENTS.

- **HELP_KEY**
  - Display the topic in the keyword table that matches the keyword(s) in the string passed as `dwData`. If multiple matches are found, display the Index topic with each found topic in the Topics Found dialog box. Multiple keywords in `dwData` must be separated by semicolons.

- **HELP_MULTIKEY**
  - Display the topic specified by a keyword in an alternative keyword table. `dwData` is a `MULTIKEYHELP` structure which specifies a table footnote character and a keyword.

- **HELP_PARTIALKEY**
  - Same as HELP_KEY, except that to display the index without passing a keyword, pass an empty string as `dwData`.

- **HELP_QUIT**
  - Close Windows Help unless other programs currently need it.

- **HELP_SETCONTENTS**
  - Set which Help topic is considered to be the Contents topic. `dwData` is the context identifer of the topic to set as the Contents.

- **HELP_SETINDEX**
  - Same as HELP_SETCONTENTS.
HELP_SETPOPUP_POS
Set the position of a subsequent pop-up window. *dwData* is a **POINT_TYPE** structure identifying the coordinates of the upper-left corner of the subsequent pop-up window.

HELP_SETWINPOS
Display the Help window, if it is minimized or hidden, and set its size and position. *dwData* is a **HELPWININFO** structure holding the size and position information of the desired Help window.

HELP_TCARD
Indicate that the topic to display is for a training card. This must be combined with another flag.

HELP_WMHELP
Display the topic for the control identified by *hWndMain*. *dwData* is an array of pairs of Longs (dwords): the first in a pair is a control identifier, and the second in a pair is the context identifier of the associated Help topic. The array's last pair must be two zeros.

*dwData*
Depends on the value of *uCommand*.

**Constant Definitions**

```
Const HELP_COMMAND = &H102
Const HELP_CONTENTS = &H3
Const HELP_CONTEXT = &H1
Const HELP_CONTEXTMENU = &HA
Const HELP_CONTEXTPOPUP = &H8
Const HELP_FINDER = &HB
Const HELP_FORCEFILE = &H9
Const HELP_HELPONHELP = &H4
Const HELP_INDEX = &H3
Const HELP_KEY = &H101
Const HELP_MULTIKEY = &H201
Const HELP_PARTIALKEY = &H105
Const HELP_QUIT = &H2
Const HELP_SETCONTENTS = &H5
Const HELP_SETINDEX = &H5
Const HELP_SETPOPUP_POS = &HD
Const HELP_SETWINPOS = &H203
Const HELP_TCARD = &H8000
Const HELP_WMHELP = &HC
```

**Example**

```
' This code is licensed according to the terms and conditions listed here.
' Display the Help Topics dialog box of the Help file C:\MyApp\appehelp.hlp.
' The window Form1 is opening Windows Help.
Dim retval As Long ' return value
```

'Display the Help file as mentioned above. Note how the ByVal keyword must 'be used because we're passing a regular Long to it.
retval = **WinHelp**(Form1.hWnd, "C:\MyApp\apphelp.hlp", HELP_FINDER, ByVal 0)

**Category**

Help

Go back to the alphabetical Function listing.
Go back to the Reference section index.

---

*Last Modified:* August 13, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information] Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/w/winhelp.html](http://www.vbapi.com/ref/w/winhelp.html)
WriteFile Function

Declare Function WriteFile Lib "kernel32.dll" (ByVal hFile As Long, lpBuffer As Any, ByVal nNumberOfBytesToWrite As Long, lpNumberOfBytesWritten As Long, lpOverlapped As Any) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

**WriteFile** writes a series of bytes (originally stored in any variable, array, or structure) to a file. The file must of course have been opened with at least write-level access. If the file is synchronous (not overlapped), the function begins writing to the file at the current position of the file pointer, and the function automatically adjusts the file pointer to point to the byte immediately after the last byte written. If the file is asynchronous (overlapped), the structure passed as *lpOverlapped* identifies the point to begin writing at, and the program calling the function is responsible for updating the file pointer.

Return Value

If an error occurred, the function returns 0 (use **GetLastError** to get the error code). If successful, the function returns a non-zero value.

Visual Basic-Specific Issues

When passing a string as *lpBuffer*, the ByVal keyword must preceed the string. The keyword is not necessary for any other data types passed for *lpBuffer*. When passing 0 for *lpOverlapped*, the expression **ByVal CLng(0)** must be used.
Parameters

hFile
The handle to the file to write to. The file must have at least write-level access.

lpBuffer
The variable, array, or string holding the data to write to the file.

nNumberOfBytesToWrite
The number of bytes of data to write to the file.

lpNumberOfBytesWritten
Receives the number of bytes of data actually written to the file.

lpOverlapped
If the file is asynchronous (overlapped), this is an OVERLAPPED structure specifying where to begin writing at. If the file is synchronous (not overlapped), this must be 0.

Example

' This code is licensed according to the terms and conditions listed here.

' Write both a Long (32-bit) number and a String to the file
' C:\Test\myfile.txt. Notice how the ByVal keyword must be used
' when writing a string variable.
Dim longbuffer As Long  ' long to write to the file
Dim stringbuffer As String  ' string to write to the file
Dim numwritten As Long  ' receives number of bytes written to the file
Dim hFile As Long  ' handle of the open file
Dim retval As Long  ' return value

' Open or create the file being written to.
hFile = CreateFile("C:\Test\myfile.txt", GENERAL_WRITE, FILE_SHARE_READ, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_ARCHIVE, 0)
If hFile = -1 Then  ' the file could not be opened
    Debug.Print "Unable to open the file -- it probably does not exist."
    End  ' abort the program
End If

' Write a Long-type number (27) to the file
longbuffer = 27  ' the Long value to write to the file
retval = WriteFile(hFile, longbuffer, Len(longbuffer), numwritten, CLng(0))

' Write a 10-character string to the file
stringbuffer = "Anonymous!"  ' the String to write to the file
retval = WriteFile(hFile, ByVal stringbuffer, 10, numwritten, CLng(0))

' Close the file.
retval = CloseHandle(hFile)

See Also

ReadFile, SetFilePointer

Category

Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: October 13, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/writefile.html
WritePrivateProfileString Function

Declare Function WritePrivateProfileString Lib "kernel32.dll" Alias "WritePrivateProfileStringA" (ByVal lpApplicationName As String, ByVal lpKeyName As String, ByVal lpString As String, ByVal lpFileName As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

WritePrivateProfileString sets a value inside of an INI file. This function can also be used to set numerical values if they are in string form, for example using "1" to represent the number 1. If the INI file you try to write to does not exist, it will be created. Likewise, if the section or value does not exist, it will also be created. The function returns 0 if an error occurs, or 1 if successful. Note that INI file support is only provided in Windows for backwards compatibility; using the registry to store information is preferred.

lpApplicationName
  The section of the INI file to write to.
lpKeyName
  The name of the value to set.
lpString
  The string to set as the value.
lpFileName
  The filename of the INI file to write to.

Example:

' Set the "username" setting in the [Default] section of
' C:\MyProgram\config.ini to "Rimmer". Also set the "useinfo" setting under the same
' section to 1 (i.e., "1").
Dim retval As Long ' return value

' Set the string value.
retval = WritePrivateProfileString("Default", "username", "Rimmer", "C:\MyPrograms\config.ini")

' Set the numeric value.
retval = WritePrivateProfileString("Default", "useinfo", "1", "C:\MyPrograms\config.ini")

See Also: GetPrivateProfileString, WriteProfileString
Category: INI Files

Go back to the alphabetical Function listing.
Go back to the Reference section index.
WriteProfileString Function

Declare Function WriteProfileString Lib "kernel32.dll" Alias "WriteProfileStringA" (ByVal lpszSection As String, ByVal lpszKeyName As String, ByVal lpszString As String) As Long

Platforms: Win 32s, Win 95/98, Win NT

WriteProfileString sets a value inside of the WIN.INI file. This function can also be used to set numerical values if they are in string form, for example using "1" to represent the number 1. If the section or value does not exist, it will be created. Note that, since Windows normally handles the WIN.INI file by itself, extreme care should be taken when writing directly to it. This function is basically a watered-down version of WritePrivateProfileString because, unlike this function, it works with any INI file. The function returns 0 if an error occurs, or a non-zero value if successful. Note that INI file support is only provided in Windows for backwards compatibility; using the registry to store information is preferred.

lpszSection
The section of WIN.INI to write to.
lpszKeyName
The name of the value to set.
lpszString
The string to set as the value.

Example:

' Set the "Wallpaper" setting in the [Desktop] section of WIN.INI
to C:\Windows\Clouds.bmp.
' WARNING: Use extreme caution when editing the WIN.INI file, because writing bad
data to
' it could have unpredictable and disastrous results to the system!
Dim retval As Long  ' return value

' Set the value.
retval = WriteProfileString("Desktop", "Wallpaper", "C:\Windows\Clouds.bmp")

See Also: GetProfileString, WritePrivateProfileString
Category: INI Files
WSACleanup Function

Declare Function WSACleanup Lib "wssock32.dll" () As Long

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

WSACleanup frees the resources allocated when when the program began using Windows Sockets (Winsock) functions. Your program must call this function once it no longer needs to use Winsock functions, so that it frees the resources the prior call to WSASStartup consumed. The resources will not be freed unless the program calls WSACleanup the same number of times as it called WSASStartup (in case that function had been called more than once).

Return Value

If successful, the function returns zero. If an error occured, the function returns -1 (use WSAGetLastError to get the error code).

Visual Basic-Specific Issues

None.

Parameters

None.

Example

Print the IP address associated with a domain name specified by the user. Winsock is briefly used to resolve the domain name and format a printable IP address string. The user enters the domain name to resolve in text box txtDomain, and the domain name is resolved when the user clicks button cmdGetIP.
Windows API Guide: WSACleanup Function

To run this example, place a text box named txtDomain and a command button named cmdGetIP on a form window.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, ByVal lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function inet_ntoa Lib "wsock32.dll" (ByVal inaddr As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

' Define a relevant API macro.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***
Private Sub cmdGetIP_Click()
    Dim sockinfo As WSADATA ' information about Winsock
    Dim hostinfo As HOSTENT ' information about an Internet host

Dim pHostinfo As Long ' pointer to a HOSTENT structure
Dim pIPAddress As Long ' pointer to an IP address dword
Dim ipAddress As Long ' an IP address, packed into a dword
Dim pIPString As Long ' pointer to an IP address formatted as a string
Dim ipString As String ' holds a human-readable IP address string
Dim retval As Long ' generic return value

' Open up a Winsock session, using version 2.2.
retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(txtDomain.Text)
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve domain name."
Else
    ' Copy the data into a HOSTENT structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print "A non-IP address was returned."
    Else
        ' Copy the pointer to the first (and probably only) IP
directory in the structure.
        CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
        ' Copy the actual IP address.
        CopyMemory ipAddress, ByVal pIPAddress, 4
        ' Convert the IP address into a human-readable string.
        pIPString = inet_ntoa(ipAddress)
        ' Copy the result into a string variable.
        ipString = Space(lstrlen(pIPString))
        retval = lstrcpy(ipString, pIPString)
        ' Print the result: a human-readable IP address.
        Debug.Print ipString
    End If
End If

' Close the Winsock session.
retval = WSACleanup()

End Sub

See Also

WSAStartup

Category
WSAGetLastError Function

Declare Function WSAGetLastError Lib "wsock32.dll" () As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

WSAGetLastError identifies the last error code reported by the last Windows Sockets (Winsock) function call. Note that successful Winsock functions typically do not clear this error code, so **WSAGetLastError** will identify the last error that occurred. Since most Winsock functions only return whether or not they failed, this function should be used to get a better description of the exact error that occurred.

Return Value

The function returns one of the [Winsock error codes](http://216.26.168.92/vbapi/ref/w/wsagetlasterror.html) identifying the last Winsock error that occurred.

Visual Basic-Specific Issues

None.

Parameters

None.

Example

Intentionally cause an error by failing to initialize Winsock before calling its functions. This is the same code that is in the [WSAStartup](http://216.26.168.92/vbapi/ref/w/wsagetlasterror.html) page's example, except that the call to WSAStartup has been removed. This code intentionally does not work. Any Winsock errors that occur should be reported via **WSAGetLastError**.

To run this example, place a text box named txtDomain and a command button named cmdGetIP on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Declare Function WSAGetLastError Lib "wsock32.dll" () As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long

Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type

Public Const AF_INET = 2
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function inet_ntoa Lib "wsock32.dll" (ByVal inaddr As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

Define some useful API macros.

Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

Public Function LOBYTE(ByVal wValue As Integer) As Byte
    LOBYTE = Val("&H" & Right("00" & Hex(wValue), 2))
End Function

Public Function HIBYTE(ByVal wValue As Integer) As Byte
    HIBYTE = Val("&H" & Left(Right("0000" & Hex(wValue), 4), 2))
End Function

In the event of an error, print the Winsock error number in the Debug window. Ideally the error code would be used in a more meaningful manner in a real program (e.g., printing out a textual error message), but this is just an example.

Public Sub ReportWinsockError ()
    Dim winsockError As Long  ' the Winsock error code
    winsockError = WSAGetLastError()
    Debug.Print "Winsock error": winsockError; "reported!"
End Sub
Private Sub cmdGetIP_Click()
    Dim sockinfo As WSADATA  ' information about Winsock
    Dim hostinfo As HOSTENT ' information about an Internet host
    Dim pHostinfo As Long   ' pointer to a HOSTENT structure
    Dim pIPAddress As Long  ' pointer to an IP address dword
    Dim ipAddress As Long   ' an IP address, packed into a dword
    Dim pIPString As Long   ' pointer to an IP address formatted as a string
    Dim ipString As String  ' holds a human-readable IP address string
    Dim retval As Long      ' generic return value

    ' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(txtDomain.Text)
    If pHostinfo = 0 Then
        Debug.Print "Unable to resolve domain name."
        ReportWinsockError
    Else
        ' Copy the data into a HOSTENT structure.
        CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
        If hostinfo.h_addrtype <> AF_INET Then
            Debug.Print "A non-IP address was returned."
            ReportWinsockError
        Else
            ' Copy the pointer to the first (and probably only) IP
            ' address in the structure.
            CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
            ' Copy the actual IP address.
            CopyMemory ipAddress, ByVal pIPAddress, 4
            ' Convert the IP address into a human-readable string.
pIPString = inet_ntoa(ipAddress)
            ' Copy the result into a string variable.
ipString = Space(lstrlen(pIPString))
            retval = lstrcpy(ipString, pIPString)
            ' Print the result: a human-readable IP address.
            Debug.Print ipString
        End If
    End If
    ' Close the Winsock session.
    retval = WSACleanup()
End Sub

Category

Winsock

Back to the Function list.
WSAStartup Function

Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

WSAStartup initializes the use of Windows Sockets (Winsock) for the calling program. This function must be called before any other Winsock functions are called, in order to perform any necessary initialization. The program can specify which version of Winsock it wants to use. When your program no longer needs to call Winsock function, it must call WSACleanup to free resources.

Return Value

If successful, the function returns zero. If an error occurred, the function returns one of the following one of the following values specifying the error:

WSAEFAULT
   A valid WSADATA structure was not passed for lpWSAData.
WSAEINPROGRESS
   A Winsock 1.1 blocking operation is in progress.
WSAEPROCLIM
   The limit of the number of tasks supported by the Winsock implementation has been reached.
WSASYSCALLNOTRETRY
   The network subsystem is not ready for network communication.
WSAVERNOTSUPPORTED
   The requested version of Winsock is not provided by the current Winsock implementation.

Visual Basic-Specific Issues

None.
Parameters

\textit{wVersionRequested}

The highest version of Winsock your program can use. If you wish, your program could specify a lower version than 2.2. The low-order byte contains the major version number, and the high-order byte contains the minor version number. In other words, &H202 means Winsock 2.2 and &H2 means Winsock 2.0.

\textit{lpWSAData}

Receives information about the current Winsock implementation.

Constant Definitions

\begin{verbatim}
Const WSAEFAULT = 10014
Const WSAEINVAL = 10036
Const WSAEPROCLIM = 10067
Const WSASYSNOTREADY = 10091
Const WSAVERNOTSUPPORTED = 10092
\end{verbatim}

Example

Print the IP address associated with a domain name specified by the user. Winsock is briefly used to resolve the domain name and format a printable IP address string. The user enters the domain name to resolve in text box \texttt{txtDomain}, and the domain name is resolved when the user clicks button \texttt{cmdGetIP}.

To run this example, place a text box named \texttt{txtDomain} and a command button named \texttt{cmdGetIP} on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the \texttt{(declarations) section of a module.)
Public Type \texttt{WSADATA}
  wVersion As Integer
  wHighVersion As Integer
  szDescription As String * 257
  szSystemStatus As String * 129
  iMaxSockets As Long
  iMaxUdpDg As Long
  lpVendorInfo As Long
End Type
Public Declare Function \texttt{WSAStartup} Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSADATA _
  As \texttt{WSADATA}) As Long
Public Declare Function \texttt{WSACleanup} Lib "wsock32.dll" () As Long
Public Type \texttt{HOSTENT}
  h_name As Long
  h_aliases As Long
  h_addrtype As Integer
  h_length As Integer
  h_addr_list As Long
\end{verbatim}
End Type
Public Const AF_INET = 2
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function inet_ntoa Lib "wsock32.dll" (ByVal inaddr As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long

' Define some useful API macros.
Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function
Public Function LOBYTE(ByVal wValue As Integer) As Byte
    LOBYTE = Val("&H" & Right("00" & Hex(wValue), 2))
End Function
Public Function HIBYTE(ByVal wValue As Integer) As Byte
    HIBYTE = Val("&H" & Left(Right("0000" & Hex(wValue), 4), 2))
End Function

' *** Place the following code inside the form window. ***
Private Sub cmdGetIP_Click()
    Dim sockinfo As WSADATA ' information about Winsock
    Dim hostinfo As HOSTENT ' information about an Internet host
    Dim pHostinfo As Long ' pointer to a HOSTENT structure
    Dim pIPAddress As Long ' pointer to an IP address dword
    Dim ipAddress As Long ' an IP address, packed into a dword
    Dim ipString As Long ' pointer to an IP address formatted as a string
    Dim ipString As String ' holds a human-readable IP address string
    Dim retval As Long ' generic return value

    ' Open up a Winsock session, using version 2.2.
    retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
    If retval <> 0 Then
        Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
        Exit Sub
    End If

    ' Display the version of Winsock being used, and the description
    ' provided by the implementation (after removing the empty space).
    Debug.Print "Using Winsock version"; LOBYTE(sockinfo.wVersion); ";";
HIBYTE(sockinfo.wVersion)
   Debug.Print "Description: "; Left(sockinfo.szDescription, _
   InStr(sockinfo.szDescription, vbNullChar) - 1)

' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(txtDomain.Text)
If pHostinfo = 0 Then
   Debug.Print "Unable to resolve domain name."
Else
   ' Copy the data into a HOSTENT structure.
   CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
   If hostinfo.h_addrtype <> AF_INET Then
      Debug.Print "A non-IP address was returned."
   Else
      ' Copy the pointer to the first (and probably only) IP
      address in the structure.
      CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
      ' Copy the actual IP address.
      CopyMemory ipAddress, ByVal pIPAddress, 4
      ' Convert the IP address into a human-readable string.
      pIPString = inet_ntoa(ipAddress)
      ' Copy the result into a string variable.
      ipString = Space(lstrlen(pIPString))
      retval = lstrcpy(ipString, pIPString)
      ' Print the result: a human-readable IP address.
      Debug.Print ipString
   End If
End If

' Close the Winsock session.
retval = WSACleanup()

End Sub

See Also

WSACleanup

Category

Winsock

Back to the Function list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
ZeroMemory Function

Declare Sub ZeroMemory Lib "kernel32.dll" Alias "RtlZeroMemory" (Destination As Any, ByVal Length As Long)

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Not Supported.

Description & Usage

ZeroMemory fills a location in memory with zeros. The function sets each byte starting at the given memory location to zero. The memory location is identified by a pointer to the memory address.

Return Value

ZeroMemory does not return a value.

Visual Basic-Specific Issues

A pointer to any variable can be automatically generated merely by passing that variable as Destination. However, if either a String or a Long holding the desired memory address is passed, the ByVal keyword must preceed it.

Parameters

- **Destination**
  
  A pointer to the location in memory (often the memory address of a variable) to begin filling with zeros.

- **Length**
  
  The number of memory bytes, beginning with the address identified by Destination, to set to zeros.

Example
This code is licensed according to the terms and conditions listed [here](#).

Initialize all the elements in an array to the value 0.
Dim bytearray(0 To 9) as Byte ' array of 10 bytes
Dim c As Integer ' counter variable

Fill the memory at bytearray() with zeros. Note that, to identify the pointer to bytearray()'s memory location, it is passed as normal.
ZeroMemory bytearray(0), 10 ' zero out 10 bytes
'Display the results to verify that it worked.
For c = 0 To 9 ' loop through each element
  Debug.Print bytearray(c); ' each value displayed will be 0
Next c

See Also

FillMemory

Category

Memory

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: July 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/z/zeromemory.html](http://www.vbapi.com/ref/z/zeromemory.html)
Windows API Reference: Functions

Last Update: January 21, 2001
Number of Functions Listed: 357 (4 added)

Below is a categorical list of the API functions currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to:

- Accessibility
- Audio
- Bitmaps
- Brushes
- Common Controls
- Common Dialog
- Cursor
- Devices
- Dialog Boxes
- Errors
- Files
- Filled Shapes
- Fonts & Text
- Handles
- Help
- Icons
- INI Files
- Input (General)
- Joysticks
- Keyboard
- Lines & Curves
Windows API Guide: Reference: Functions

- Math
- Media Control Interface (MCI)
- Memory
- Menus
- Messages
- Mouse
- National Language Support
- OLE
- Painting & Drawing
- Pens
- Printers
- Processes & Threads
- Rectangles
- Regions
- Registry
- Shell
- Shutdown
- Strings
- Synchronization
- System Information
- Time
- Timers
- Tool Help
- Window Classes
- Window Procedures
- Window Properties
- Windows
- Winsock
- Other

- Accessibility
  - GetSystemMetrics
  - SystemParametersInfo

- Audio
  - auxGetDevCaps
  - auxGetNumDevs
  - auxGetVolume
Windows API Guide: Reference: Functions

- auxSetVolume
- PlaySound
- sndPlaySound
- waveOutGetDevCaps
- waveOutGetNumDevs
- waveOutGetVolume
- waveOutSetVolume

- **Bitmaps**
  - BitBlt
  - ExtFloodFill
  - GetPixel
  - SetPixel
  - SetPixelV
  - StretchBlt

- **Brushes**
  - CreateHatchBrush
  - CreateSolidBrush
  - GetBrushOrgEx
  - SetBrushOrgEx

- **Common Controls**
  - InitCommonControlsEx

- **Common Dialog**
  - ChooseColor
  - ChooseFont
  - CommDlgExtendedError
  - GetOpenFileName
  - GetSaveFileName
  - PrintDlg

- **Cursor**
  - ClipCursor
  - CreateCursor
  - DestroyCursor
  - GetClipCursor
  - GetCursor
  - GetCursorPos
  - LoadCursor
  - LoadCursorFromFile
  - SetCursor
  - SetCursorPos
Windows API Guide: Reference: Functions

- **SetSystemCursor**
- **ShowCursor**

**Devices**
- **ChangeDisplaySettings** **NEW**
- **CreateDC**
- **DeleteDC**
- **DeleteObject**
- **EnumDisplaySettings** **NEW**
- **GetDC**
- **GetStockObject**
- **ReleaseDC**
- **SelectObject**

**Dialog Boxes**
- **MessageBox**
- **MessageBoxEx**
- **MessageBoxIndirect**

**Errors**
- **Beep**
- **GetLastError**
- **MessageBeep**
- **SetLastError**
- **SetLastErrorEx**

**Files**
- **CopyFile**
- **CreateDirectory**
- **CreateDirectoryEx**
- **CreateFile**
- **DeleteFile**
- **FindClose**
- **FindFirstFile**
- **FindNextFile**
- **GetDiskFreeSpace**
- **GetDiskFreeSpaceEx**
- **GetDriveType**
- **GetFileAttributes**
- **GetFileInformationByHandle**
- **GetFileSize**
- **GetFileTime**
- **GetFileVersionInfo**
- **GetFileVersionInfoEx**
Windows API Guide: Reference: Functions

- GetFileVersionInfoSize
- GetFullPathName
- GetLogicalDrives
- GetLogicalDriveStrings
- GetShortPathName
- GetTempFileName
- MoveFile
- ReadFile
- RemoveDirectory
- SetFileAttributes
- SetFilePointer
- SetFileTime
- VerQueryValue
- WriteFile

**File System**
- GetVolumeInformation
- SetVolumeLabel

**Filled Shapes**
- Chord
- Ellipse
- FillRect
- FrameRect
- InvertRect
- Pie
- Polygon
- PolyPolygon
- Rectangle
- RoundRect

**Fonts & Text**
- CreateFont
- CreateFontIndirect
- EnumFontFamilies
- EnumFontFamiliesEx
- GetTextAlign
- SetTextAlign
- TextOut

**Handles**
- CloseHandle

**Help**
- **WinHelp**

- **Icons**
  - **DestroyIcon**
  - **DrawIcon**
  - **DrawIconEx**
  - **ExtractIcon**
  - **ExtractIconEx**

- **INI Files**
  - **GetPrivateProfileInt**
  - **GetPrivateProfileString**
  - **GetProfileInt**
  - **GetProfileString**
  - **WritePrivateProfileString**
  - **WriteProfileString**

- **Input (General)**
  - **SendInput**

- **Joysticks**
  - **joyGetDevCaps**
  - **joyGetNumDevs**
  - **joyGetPos**

- **Keyboard**
  - **GetAsyncKeyState**
  - **GetKeyboardState**
  - **GetKeyState**
  - **keybd_event**
  - **SetKeyboardState**

- **Lines & Curves**
  - **AngleArc**
  - **Arc**
  - **ArcTo**
  - **GetArcDirection**
  - **LineTo**
  - **MoveToEx**
  - **PolyBezier**
  - **PolyBezierTo**
  - **Polyline**
  - **PolylineTo**
  - **PolyPolyline**
  - **SetArcDirection**
- **Math**
- **Media Control Interface (MCI)**
  - [mciGetErrorString](http://216.26.168.92/vbapi/ref/funcc.html)
  - [mciSendString](http://216.26.168.92/vbapi/ref/funcc.html)
- **Memory**
  - [CopyMemory](http://216.26.168.92/vbapi/ref/funcc.html)
  - [FillMemory](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GlobalMemoryStatus](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GlobalMemoryStatusEx](http://216.26.168.92/vbapi/ref/funcc.html)
  - [MoveMemory](http://216.26.168.92/vbapi/ref/funcc.html)
  - [ZeroMemory](http://216.26.168.92/vbapi/ref/funcc.html)
- **Menus**
  - [CreatePopupMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [DestroyMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GetMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GetMenuItemCount](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GetMenuItemInfo](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GetSystemMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [InsertMenuItem](http://216.26.168.92/vbapi/ref/funcc.html)
  - [RemoveMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [SetMenuItemInfo](http://216.26.168.92/vbapi/ref/funcc.html)
  - [TrackPopupMenu](http://216.26.168.92/vbapi/ref/funcc.html)
  - [TrackPopupMenuEx](http://216.26.168.92/vbapi/ref/funcc.html)
- **Messages**
  - [SendMessage](http://216.26.168.92/vbapi/ref/funcc.html)
- **Mouse**
  - [GetCapture](http://216.26.168.92/vbapi/ref/funcc.html)
  - [GetDoubleClickTime](http://216.26.168.92/vbapi/ref/funcc.html)
  - [mouse_event](http://216.26.168.92/vbapi/ref/funcc.html)
  - [ReleaseCapture](http://216.26.168.92/vbapi/ref/funcc.html)
  - [SetCapture](http://216.26.168.92/vbapi/ref/funcc.html)
  - [SetDoubleClickTime](http://216.26.168.92/vbapi/ref/funcc.html)
  - [SwapMouseButton](http://216.26.168.92/vbapi/ref/funcc.html)
- **National Language Support**
- GetCurrencyFormat
- GetDateFormat
- GetNumberFormat
- GetThreadLocale
- GetTimeFormat
- SetThreadLocale

- OLE
  - CoTaskMemFree

- Painting & Drawing
  - GetWindowRgn
  - SetWindowRgn

- Pens
  - CreatePen
  - CreatePenIndirect

- Printers
  - ClosePrinter
  - EndDoc
  - EndPage
  - EnumJobs
  - EnumPrinters
  - OpenPrinter
  - PrinterProperties NEW
  - StartDoc
  - StartPage

- Processes & Threads
  - GetEnvironmentVariable
  - SetEnvironmentVariable

- Rectangles
  - CopyRect
  - EqualRect
  - InflateRect
  - IntersectRect
  - IsRectEmpty
  - OffsetRect
  - PtInRect
  - Rect
  - RectEmpty
  - SubtractRect
  - UnionRect
- **Regions**
  - CombineRgn
  - CreateEllipticRgn
  - CreateEllipticRgnIndirect
  - CreatePolygonRgn
  - CreatePolyPolygonRgn
  - CreateRectRgn
  - CreateRectRgnIndirect
  - CreateRoundRectRgn
  - EqualRgn
  - FillRgn
  - FrameRgn
  - GetPolyFillMode
  - GetRgnBox
  - InvertRgn
  - OffsetRgn
  - PtInRegion
  - RectInRegion
  - SetPolyFillMode

- **Registry**
  - RegCloseKey
  - RegCreateKeyEx
  - RegDeleteKey
  - RegDeleteValue
  - RegEnumKeyEx
  - RegEnumValue
  - RegOpenKeyEx
  - RegQueryValueEx
  - RegSetValueEx

- **Shell**
  - ExitWindowsDialog
  - PickIconDlg
  - RestartDialog
  - SHAddToRecentDocs
  - SHBrowseForFolder
  - Shell_NotifyIcon
  - ShellExecute
  - ShellExecuteEx
  - SHEmptyRecycleBin
- SHFileOperation
- SHFreeNameMappings
- SHGetFileInfo
- SHGetFolderLocation
- SHGetFolderPath
- SHGetPathFromIDList
- SHGetSpecialFolderLocation
- SHGetSpecialFolderPath
- SHQueryRecycleBin
- SHUpdateRecycleBinIcon

- **Shutdown**
  - LockWorkStation

- **Strings**
  - CharLower
  - CharUpper
  - CompareString
  - lstr cmp
  - lstr cmp i
  - lstr cp y
  - lstr cpyn
  - lstrlen

- **Synchronization**
  - WaitForSingleObject

- **System Information**
  - GetComputerName
  - GetSysColor
  - GetSystemDirectory
  - GetTempPath
  - Get UserName
  - GetVersionEx
  - GetWindowsDirectory
  - SetSysColors

- **Time**
  - CompareFileTime
  - File Time To Local File Time
  - File Time To System Time
  - Get Local Time
  - Get System Time
  - Get System Time As File Time
- **GetTickCount**
- **GetTimeZoneInformation**
- **LocalFileTimeToFileTime**
- **SetSystemTime**
- **SystemTimeToFileTime**

**Timers**
- **KillTimer**
- **QueryPerformanceCounter**
- **QueryPerformanceFrequency**
- **SetTimer**

**Tool Help**
- **CreateToolhelp32Snapshot**
- **Process32First**
- **Process32Next**

**Window Classes**
- **GetClassInfo**
- **GetClassInfoEx**
- **GetClassLong**
- **GetClassName**
- **GetWindowLong**
- **RegisterClass**
- **RegisterClassEx**
- **SetClassLong**
- **SetWindowLong**
- **UnregisterClass**

**Window Procedures**
- **CallWindowProc**
- **DefWindowProc**

**Window Properties**
- **EnumPropsEx**
- **GetProp**
- **RemoveProp**
- **SetProp**

**Windows**
- **BringWindowToTop**
- **CreateWindowEx**
- **DestroyWindow**
- **EnableWindow**
- **EnumChildWindows**
Windows API Guide: Reference: Functions

- **EnumThreadWindows**
- **EnumWindows**
- **FindWindow**
- **FindWindowEx**
- **FlashWindow**
- **GetActiveWindow**
- **GetDesktopWindow**
- **GetFocus**
- **GetForegroundWindow**
- **GetParent**
- **GetTopWindow**
- **GetWindow**
- **GetWindowRect**
- **GetWindowText**
- **GetWindowTextLength**
- **GetWindowThreadProcessId**
- **IsChild**
- **IsIconic**
- **IsWindow**
- **IsWindowEnabled**
- **IsZoomed**
- **MoveWindow**
- **SetActiveWindow**
- **SetFocus**
- **SetForegroundWindow**
- **SetParent**
- **SetWindowPos**
- **SetWindowText**
- **ShowWindow**
- **WindowFromPoint**

- **Winsock**
  - **closesocket**
  - **connect**
  - **gethostbyaddr**
  - **gethostbyname**
  - **gethostname**
  - **htonl**
  - **htons**
  - **inet_addr**
Windows API Guide: Reference: Functions

- inet_nota
- ioctlsocket **NEW**
- recv
- send
- socket
- WSACleanup
- WSAGetLastError
- WSAStartup

- **Other**
  - ExitWindowsEx
  - Sleep

Go back to the Reference section index.

_Last Modified:_ January 21, 2001

This page is copyright © 2001 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/copyright.html) [Revised October 29, 2000](http://216.26.168.92/vbapi/ref/revision.html)

Go back to the [Windows API Guide home page](http://www.vbapi.com/ref/home.html).

E-mail: [vbapi@vbapi.com](mailto:vbapi@vbapi.com) [Send Encrypted E-Mail](http://www.vbapi.com/ref/encrypted.html)

This page is at [http://www.vbapi.com/ref/funcc.html](http://www.vbapi.com/ref/funcc.html)
Windows API Reference: Structures

Last Update: January 21, 2001
Number of Structures Listed: 83 (0 added)

Below is an alphabetical list of the API structures currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to: A | B | C | D | E | F | G | H | I | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

- A
  - ACCESSTIMEOUT
  - ACL
  - ANIMATIONINFO
  - AUXCAPS

- B
  - BROWSEINFO
  - BY_HANDLE_FILE_INFORMATION

- C
  - CHOOSECOLOR_TYPE
  - CHOOSEFONT_TYPE
  - CURRENCYFMT

- D
  - DEVMODE
  - DEVNAMES
  - DOCINFO

- E
  - ENUMLOGFONT
  - ENUMLOGFONTEX
- **F**
  - FILETIME
  - FILTERKEYS
  - FONTSIGNATURE

- **G**

- **H**
  - HARDWAREINPUT
  - HELPINFO
  - HELPWININFO
  - HIGHCONTRAST
  - HOSTENT

- **I**
  - ICONMETRICS
  - INITCOMMONCONTROLSEX_TYPE
  - INPUT_TYPE
  - ITEMIDLIST

- **J**
  - JOB_INFO_1
  - JOB_INFO_2
  - JOYCAPS
  - JOYINFO

- **K**
  - KEYBDINPUT

- **L**
  - LARGE_INTEGER
  - LOGFONT
  - LOGPEN

- **M**
  - MEMORYSTATUS
  - MEMORYSTATUSEX
  - MNUITEMINFO
  - MINIMIZEDMETRICS
  - MOUSEINPUT
  - MOUSEKEYS
  - MSGBOXPARAMS
  - MULTIKEYHELP

- **N**
  - NEWTEXTMETRIC
  - NEWTEXTMETRICEX
- NONCLIENTMETRICS
- NOTIFYICONDATA
- NUMBERFMT

- O
  - OPENFILENAME
  - OSVERSIONINFO
  - OVERLAPPED

- P
  - POINT_TYPE
  - PRINTDLG_TYPE
  - PRINTER_DEFAULTS
  - PRINTER_INFO_1
  - PRINTER_INFO_2
  - PRINTER_INFO_4
  - PRINTER_INFO_5
  - PROCESSENTRY32

- Q

- R
  - RECT

- S
  - SECURITY_ATTRIBUTES
  - SECURITY_DESCRIPTOR
  - SERIALKEYS
  - SHELLEXECUTEINFO
  - SHFILEINFO
  - SHFILEOPSTRUCT
  - SHITEMID
  - SHNAMEMAPPING
  - SHQUERYRBINFO
  - SOCKADDR
  - SOUNDSENTRY
  - STICKYKEYS
  - SYSTEMTIME

- T
  - TEXTMETRIC
  - TIME_ZONE_INFORMATION
  - TOGGLEKEYS
  - TPMPARAMS

- U
- ULARGE_INTEGER
- VS_FIXEDFILEINFO
- WAVEOUTCAPS
- WIN32_FIND_DATA
- WNDCLASS
- WNDCLASSEX
- WSADATA
- X
- Y
- Z

Go back to the Reference section index.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/structa.html
ACCESTIMEOUT Structure

Type ACCESTIMEOUT
    cbSize As Long
    dwFlags As Long
    iTimeOutMSec As Long
End Type

ACCESTIMEOUT-type variables specify information about the time-out feature of the Windows accessibility features. After the time-out period elapses (beginning with the last user input), the FilterKeys, HighContrast, MouseKeys, StickyKeys, and ToggleKeys accessibility features are disabled. The structure specifies settings for the time-out feature.

cbSize
The size in bytes of the structure.
dwFlags
Zero or more of the following flags specifying the settings of the time-out feature:
ATF_AVAILABLE = &H4
    The time-out period can be changed (this flag can be read but not set).
ATF_ONOFFFEEDBACK = &H2
    Play a sound when the time-out period elapses and the accessibility features are deactivated.
ATF_TIMEOUTON = &H1
    The time-out feature is activated.
iTimeOutMSec
The number of milliseconds after the last mouse or keyboard input to wait (the time-out period) before deactivating the accessibility features.

Used by: SystemParametersInfo

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
ACL Structure

Type ACL
    AclRevision As Byte
    Sbz1 As Byte
    AclSize As Integer
    AceCount As Integer
    Sbz2 As Integer
End Type

ACL-type variables store information about an access-control list (ACL). The ACL structure is followed by zero or more access-control entries (ACEs) which the ACL is made up of. Note how the ACEs are not actually stored inside the ACL structure.

AclRevision
    Must be set to the following flag which specifies the ACL's revision level:
    ACL_REVISION = 2
    The only valid revision level.

Sbz1
    Reserved -- set to 0. This member merely aligns the structure's other members in memory.

AclSize
    The combined size in bytes of this structure and all of the ACEs which follow it.

AceCount
    The number of ACEs which follow this structure.

Sbz2
    Reserved -- set to 0. This member merely aligns the structure's other members in memory.

Used by: SECURITY_DESCRIPTOR

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
ANIMATIONINFO Structure

Type ANIMATIONINFO
    cbSize As Long
    iMinAnimate As Long
End Type

ANIMATIONINFO-type variables identify the settings of animation effects in Windows. The structure specifies whether Windows displays the animation when maximizing, restoring, or minimizing a window.

(cbSize)
    The size in bytes of the structure.

(iMinAnimate)
    If 0, then do not display the animations when maximizing, restoring, or minimizing a window. If non-zero, then display the animations.

Used by: SystemParametersInfo

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/(?/name).html
### AUXCAPS Structure

**Type** AUXCAPS  

```vbnet
Type AUXCAPS  
wMid As Integer  
wPid As Integer  
vDriverVersion As Long  
szPname As String * 32  
wTechnology As Integer  
wReserved1 As Integer  
dwSupport As Long  
End Type
```

### Description & Usage

The **AUXCAPS** structure stores information about an auxiliary audio device's capabilities. The structure also contains identifying information about the device, such as its version number.

### Visual Basic-Specific Issues

None.

### Data Members

- **wMid**  
  The *manufacturer identifier* of the device.

- **wPid**  
  The *product identifier* of the device.

- **vDriverVersion**  
  The version number of the device driver. The high-order byte contains the major version number, and the low-order byte contains the minor version number.

- **szPname**  
  The name of the device, terminated by a null character.

- **wTechnology**  
  The *device's* capabilities.
One of the following flags specifying which type of auxiliary audio device it is:

- **AUXCAPS_AUXIN**
  - Line input audio device.
- **AUXCAPS_CDAUDIO**
  - CD audio device.
- **AUXCAPS_MASTER**
  - Master audio device.
- **AUXCAPS_MIC**
  - Microphone audio device.
- **AUXCAPS_MIDI**
  - MIDI audio device.
- **AUXCAPS_PCSPEAKER**
  - PC speaker audio device.
- **AUXCAPS_WAVE**
  - Waveform audio device.

**wReserved1**

- Reserved -- set to 0.

**dwSupport**

A combination of zero or more of the following flags specifying various features which the device supports:

- **AUXCAPS_LRVOLUME**
  - Supports separate left and right channel volume control.
- **AUXCAPS_VOLUME**
  - Supports volume control.

### Constant Definitions

- Const **AUXCAPS_AUXIN** = &H2
- Const **AUXCAPS_CDAUDIO** = &H1
- Const **AUXCAPS_MASTER** = &H8
- Const **AUXCAPS_MIC** = &H4
- Const **AUXCAPS_MIDI** = &H40
- Const **AUXCAPS_PCSPEAKER** = &H10
- Const **AUXCAPS_WAVE** = &H20
- Const **AUXCAPS_LRVOLUME** = &H2
- Const **AUXCAPS_VOLUME** = &H1

### Used By

- auxGetDevCaps
BROWSEINFO Structure

Type BROWSEINFO
    hwndOwner As Long
    pidlRoot As Long
    pszDisplayName As String
    lpszTitle As String
    ulFlags As Long
    lpfn As Long
    lParam As Long
    iImage As Long
End Type

Description & Usage

The BROWSEINFO structure holds information used to create the Windows shell's Browse for Folder dialog box. This structure also receives some information regarding the user's final selection.

Visual Basic-Specific Issues

Because of a quirk in Visual Basic syntax, the AddressOf operator cannot be used directly to set the lpfn parameter to a pointer to a callback function. Since the AddressOf operator is only legal within a function call, a "dummy function" must be defined by the program which simply returns the value passed to it. See the example for SHBrowseForFolder to see how this can be done.

Data Members

hwndOwner
    A handle to the window opening the dialog box.

pidlRoot
    A pointer to an ITEMIDLIST structure (a.k.a. a PIDL) which identifies the root folder for the dialog box. The user's selection is limited to this folder and any subfolders under it.

pszDisplayName
Receives the null-terminated display name of the folder the user selects. This must be initialized to an empty string of at least 260 characters.

\textit{lpszTitle}

The title of the dialog box, which will appear above the folder tree.

\textit{ulFlags}

A combination of the following flags specifying additional options for the dialog box:

- **BIF_BROWSEFORCOMPUTER**
  
  Only allow the user to select a computer.

- **BIF_BROWSEFORPRINTER**
  
  Only allow the user to select a printer.

- **BIF_BROWSEINCLUDEFILES**
  
  **\textit{With Internet Explorer 4.0 or later installed}**: Display files as well as folders in the tree.

- **BIF_DONTGOBELOWDOMAIN**
  
  Do not display network folders below the current domain level in the folder tree.

- **BIF_EDITBOX**
  
  **\textit{With Internet Explorer 4.0 or later installed}**: Display an edit box above the folder tree allowing the user to directly enter the path of the folder to select.

- **BIF_RETURNFSANCESTORS**
  
  Only allow the user to select a file system ancestor.

- **BIF_RETURNONLYFSDIRS**
  
  Only allow the user to select a file system directory.

- **BIF_STATUSTEXT**
  
  Display an area for status text, which can be set by the callback function, above the folder tree.

- **BIF_USENEWUI**
  
  **\textit{Windows 2000}**: Use the new user interface design of the dialog box. The new design includes a larger, resizable dialog box; drag and drop capability; reordering; context menus; and new folder creation, deletion, and other context menu commands.

- **BIF_VALIDATE**
  
  **\textit{With Internet Explorer 4.0 or later installed}**: If the user types an invalid path into the edit box, send the BFFM_VALIDATEFAILED message to the callback function so it can alert the user.

\textit{lpfn}

A pointer to the \textbf{BrowseCallbackProc} callback function used to process the dialog box's messages. To use the default behavior, set this to 0.

\textit{lpParam}

An application-defined value to pass to the callback function, if needed.

\textit{iImage}

Receives the index of the system image associated with the user's selection.

\section*{Constant Definitions}
Const BIF_BROWSEFORCOMPUTER = &H1000
Const BIF_BROWSEFORPRINTER = &H2000
Const BIF_BROWSEINCLUDEFILES = &H4000
Const BIF_DONTGOBELOWDOMAIN = &H2
Const BIF_EDITBOX = &H10
Const BIF_RETURNFSANCESTORS = &H8
Const BIF_RETURNONLYFSDIRS = &H1
Const BIF_STATUSTEXT = &H4
Const BIF_USENEWUI = &H40
Const BIF_VALIDATE = &H20

Used By

SHBrowseForFolder

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
BY_HANDLE_FILE_INFORMATION Structure

Type BY_HANDLE_FILE_INFORMATION
  dwFileAttributes As Long
  ftCreationTime As FILETIME
  ftLastAccessTime As FILETIME
  ftLastWriteTime As FILETIME
  dwVolumeSerialNumber As Long
  nFileSizeHigh As Long
  nFileSizeLow As Long
  nNumberOfLinks As Long
  nFileIndexHigh As Long
  nFileIndexLow As Long
End Type

BY_HANDLE_FILE_INFORMATION-type variables hold various pieces of information about a file. This information includes the file's attributes; its creation, last-access, and last-modified times and dates; the serial number of the disk the file is on; the file's size; the number of links to the file in the file system; and the unique file identifier value. Notice how the file size and file index, both being 64-bit values, are split into high-order and low-order halves of 32 bits each. To get the value they represent, you can append the hexadecimal or binary values of the two halves together. You can also use the formula  

\[
\text{actualvalue} = \text{high\_order} \times 2^{32} + \text{low\_order}
\]

\n
to calculate it.

\[ dwFileAttributes \]

One or more of the following flags which specify the file's various attributes:

\[ \text{FILE\_ATTRIBUTE\_ARCHIVE} = \&H20 \]
An archive file (which most files are).

\[ \text{FILE\_ATTRIBUTE\_COMPRESSED} = \&H800 \]
A file residing in a compressed drive or directory.

\[ \text{FILE\_ATTRIBUTE\_DIRECTORY} = \&H10 \]
A directory instead of a file.

\[ \text{FILE\_ATTRIBUTE\_HIDDEN} = \&H2 \]
A hidden file, not normally visible to the user.
FILE_ATTRIBUTE_NORMAL = &H80
    An attribute-less file (cannot be combined with other attributes).
FILE_ATTRIBUTE_READONLY = &H1
    A read-only file.
FILE_ATTRIBUTE_SYSTEM = &H4
    A system file, used exclusively by the operating system.

ftCreationTime
    The time and date of when the file was created.
ftLastAccessTime
    The time and date of when the file was last accessed.
ftLastWriteTime
    The time and date of when the file was last modified or written to.
dwVolumeSerialNumber
    The serial number of the disk which the file is stored on.
nFileSizeHigh
    The high-order half of the file's size.
nFileSizeLow
    The low-order half of the file's size.
nNumberOfLinks
    The number of links to the file in the file system. In NTFS (Win NT File System), this can be
    greater than one. In the FAT or FAT32 system (Win 32s, Win 95/98), this will always be 1.
nFileIndexHigh
    The high-order half of a unique 64-bit identifier of the file.
nFileIndexLow
    The low-order half of a unique 64-bit identifier of the file.

**Used by:** GetFileInformationByHandle

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
CHOOSECOLOR_TYPE Structure

Type CHOOSECOLOR_TYPE
  lStructSize As Long
  hwndOwner As Long
  hInstance As Long
  rgbResult As Long
  lpCustColors As Long
  Flags As Long
  lCustData As Long
  lpfnHook As Long
  lpTemplateName As String
End Type

Description & Usage

The CHOOSECOLOR_TYPE structure stores the information passed to and from the Choose Color common dialog box. The structure's data members specify both the user's selection(s) as well as other information specifying how to create the Choose Color box.

Visual Basic-Specific Issues

Officially, this structure is called CHOOSECOLOR. However, that violates the case-sensitive name spacing of Visual Basic because Visual Basic cannot then distinguish it from the ChooseColor API function. The Windows API Guide calls this structure CHOOSECOLOR_TYPE to avoid the naming collision.

Data Members

lStructSize
  The size in bytes of the structure.

hwndOwner
  A handle to the window opening the ChooseColor box, if any.
**hInstance**
A handle to a dialog template to use in place of the default box. If this is not being used, set to 0.

**rgbResult**
Set to the RGB value of the default selected color before calling ChooseColor. The function places the RGB value of the color the user selected into this member.

**lpCustColors**
A pointer to the memory block which holds the list of 16 custom colors.

**Flags**
A combination of the following flags specifying how to create the Choose Color box:
- `CC_ANYCOLOR = &H100`
  - Allow the user to select any color.
- `CC_ENABLEHOOK = &H10`
  - Use the hook function specified by lpfnHook to process the Choose Color box's messages.
- `CC_ENABLETEMPLATE = &H20`
  - Use the dialog box template identified by hInstance and lpTemplateName.
- `CC_ENABLETEMPLATEHANDLE = &H40`
  - Use the preloaded dialog box template identified by hInstance, ignoring lpTemplateName.
- `CC_FULLOPEN = &H2`
  - Automatically display the Define Custom Colors half of the dialog box.
- `CC_PREVENTFULLOPEN = &H4`
  - Disable the button that displays the Define Custom Colors half of the dialog box.
- `CC_RGBINIT = &H1`
  - Make the color specified by rgbResult be the initially selected color.
- `CC_SHOWHELP = &H8`
  - Display the Help button.
- `CC_SOLIDCOLOR = &H80`
  - Only allow the user to select solid colors. If the user attempts to select a non-solid color, convert it to the closest solid color.

**lCustData**
Application-defined value to pass to the hook function specified by lpfnHook whenever it is called.

**lpfnHook**
A pointer to the CCHookProc hook function to use to process the Choose Color box's messages. To have the dialog box process its own messages, set this to 0.

**lpTemplateName**
The name of the dialog box template to use in the module identified by hInstance. If this is not needed, set this to 0.

### Constant Definitions

```plaintext
Const CC_ANYCOLOR = &H100
Const CC_ENABLEHOOK = &H10
```
Const CC_ENABLETEMPLATE = &H20
Const CC_ENABLETEMPLATEHANDLE = &H40
Const CCFULLOPEN = &H2
Const CC_PREVENTFULLOPEN = &H4
Const CC_RGBINIT = &H1
Const CC_SHOWHELP = &H8
Const CC_SOLIDCOLOR = &H80

**Used By**

[ChooseColor](#)

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** August 14, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) **Revised October 29, 2000**
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/c/choosecolor_type.html](http://www.vbapi.com/ref/c/choosecolor_type.html)
CHOOSSEFONT_TYPE Structure

Type CHOOSSEFONT_TYPE
    lStructSize As Long
    hwndOwner As Long
    hDC As Long
    lpLogFont As Long
    iPointSize As Long
    Flags As Long
    rgbColors As Long
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
    hInstance As Long
    lpszStyle As String
    nFontType As Integer
    MISSING_ALIGNMENT As Integer
    nSizeMin As Long
    nSizeMax As Long
End Type

Description & Usage

The CHOOSSEFONT_TYPE structure is designed to pass information to and from the Choose Font common dialog box. Its members both specify initialization settings for rendering the box and receive information about the user's selection after the function completes.

Visual Basic-Specific Issues

Officially, this structure is called CHOOSSEFONT. However, that violates the case-sensitive name spacing of Visual Basic because Visual Basic cannot then distinguish it from the ChooseFont API function. The Windows API Guide calls this structure CHOOSSEFONT_TYPE to avoid the naming collision.
Data Members

lStructSize
The size in bytes of this structure.

hwndOwner
A handle to the window which is opening the Choose Font common dialog box, if any.

hdc
A handle to a device context or information context to the printer to read the printer fonts of. This must be specified if you want to display printer fonts in the font list.

lpLogFont
A pointer to a memory block which receives the contents of a LOGFONT structure specifying the attributes of the font the user selected. Optionally, this data block can also be initialized with the font to select by default in the dialog box. See the example for ChooseFont for how to obtain this address to a memory block.

iPointSize
The point size of the font, measured in units of 1/10 of a point (for example, 120 means 12 point).

Flags
A combination of the following flags specifying options for creating the Choose Font dialog box:

CF_ANSIONLY
List all fonts using a Windows or Unicode character set. Note: This flag is obsolete; use CF_SCRIPTSONLY instead.

CF_APPLY
Display and enable the Apply button.

CF_BOTH
List all printer and screen fonts.

CF_EFFECTS
Allow the strikeout, underline, and color attributes to be set.

CF_ENABLEHOOK
Use the hook function specified by lpfnHook to process the Choose Font dialog's messages.

CF_ENABLETEMPLATE
Use the dialog box template specified by lpTemplateName.

CF_ENABLETEMPLATEHANDLE
Use the preloaded dialog box template specified by hInstance.

CF_FIXEDPITCHONLY
List only fixed-pitch fonts.

CF_FORCEFONTEXIST
Do not allow the user to select a non-listed font.

CF_INITTOLOGFONTSTRUCT
Use the settings specified in lpLogFont to select a default font in the dialog box.

CF_LIMITSIZE
Limit the point size selection to values between \textit{nSizeMin} and \textit{nSizeMax} inclusive.

\begin{itemize}
  \item \textbf{CF\_NOOEMFONTS}
    \begin{itemize}
      \item Same as CF\_NOVECTORFONTS.
    \end{itemize}
  \item \textbf{CF\_NOFACESEL}
    \begin{itemize}
      \item Do not select a default font face name for the user.
    \end{itemize}
  \item \textbf{CF\_NOSCRIPTSEL}
    \begin{itemize}
      \item Do not select a default script setting for the user.
    \end{itemize}
  \item \textbf{CF\_NOSIZESEL}
    \begin{itemize}
      \item Do not select a default point size for the user.
    \end{itemize}
  \item \textbf{CF\_NOSIMULATIONS}
    \begin{itemize}
      \item Do not display a sample of the selected font.
    \end{itemize}
  \item \textbf{CF\_NOSTYLESEL}
    \begin{itemize}
      \item Do not select a default style for the user.
    \end{itemize}
  \item \textbf{CF\_NOVECTORFONTS}
    \begin{itemize}
      \item Do not list vector fonts.
    \end{itemize}
  \item \textbf{CF\_NOVERTFONTS}
    \begin{itemize}
      \item Do not list vertically-oriented fonts.
    \end{itemize}
  \item \textbf{CF\_PRINTERFONTS}
    \begin{itemize}
      \item List printer fonts.
    \end{itemize}
  \item \textbf{CF\_SCALABLEONLY}
    \begin{itemize}
      \item Only list scalable fonts.
    \end{itemize}
  \item \textbf{CF\_SCREENFONTS}
    \begin{itemize}
      \item List screen fonts.
    \end{itemize}
  \item \textbf{CF\_SCRIPTSONLY}
    \begin{itemize}
      \item List all fonts using a Windows or Unicode character set.
    \end{itemize}
  \item \textbf{CF\_SELECTSCRIPT}
    \begin{itemize}
      \item Only list fonts with the proper character set.
    \end{itemize}
  \item \textbf{CF\_SHOWHELP}
    \begin{itemize}
      \item Display the Help button.
    \end{itemize}
  \item \textbf{CF\_TTONLY}
    \begin{itemize}
      \item Only list TrueType fonts.
    \end{itemize}
  \item \textbf{CF\_USESTYLE}
    \begin{itemize}
      \item Use information in \textit{lpStyle} to initialize the dialog box.
    \end{itemize}
  \item \textbf{CF\_WYSIWYG}
    \begin{itemize}
      \item List only fonts common to the printer and the screen (must be used with CF\_BOTH and CF\_SCALABLEONLY).
    \end{itemize}
\end{itemize}

\textit{rgbColors}

The RGB value for the color of the font.

\textit{lCustData}

An application-defined parameter to pass to the hook function specified by \textit{lpfnHook}.

\textit{lpfnHook}

A pointer to a \textbf{CFHookProc} hook function used to process the dialog box's messages, if needed.

\textit{lpTemplateName}
The name of the dialog box template to use, if needed.

\textit{hInstance}

A handle to the instance of the program that contains the pre-loaded dialog box template to use, if needed.

\textit{lpszStyle}

Receives a string specifying the selected font's style settings. If used, this string must be large enough to receive the returned string.

\textit{nFontType}

One or more of the following flags specifying the type of font that is selected:

- \texttt{BOLD_FONTTYPE}
  - Boldface font.
- \texttt{ITALIC_FONTTYPE}
  - Italicized font.
- \texttt{PRINTER_FONTTYPE}
  - Printer font.
- \texttt{REGULAR_FONTTYPE}
  - Regular font -- i.e., not boldface.
- \texttt{SCREEN_FONTTYPE}
  - Screen font.
- \texttt{SIMULATED_FONTTYPE}
  - Font that can be simulated in the dialog box.

\textit{MISSING_ALIGNMENT}

Never set this variable. It is there only to align the other members of the structure in memory.

\textit{nSizeMin}

The minimum allowable point size selection, if applicable.

\textit{nSizeMax}

The maximum allowable point size selection, if applicable.

\section*{Constant Definitions}

\begin{verbatim}
Const CF_ANSIONLY = &H400
Const CF_APPLY = &H200
Const CF_BOTH = &H3
Const CF_EFFECTS = &H100
Const CF_ENABLEHOOK = &H8
Const CF_ENABLETEMPLATE = &H10
Const CF_ENABLETEMPLATEHANDLE = &H20
Const CF_FIXEDPITCHONLY = &H4000
Const CF_FORCEFONTEXIST = &H10000
Const CF_INITTOLOGFONTSTRUCT = &H40
Const CF_LIMITSIZE = &H2000
Const CF_NOOEMFONTS = &H800
\end{verbatim}
Const CF_NOFACESEL = &H80000
Const CF_NOSCRIPTSEL = &H800000
Const CF_NOSIZESEL = &H200000
Const CF_NOSIMULATIONS = &H1000
Const CF_NOSTYLESEL = &H100000
Const CF_NOVECTORFONTS = &H800
Const CF_NOVERTFONTS = &H1000000
Const CF_PRINTERFONTS = &H2
Const CF_SCALABLEONLY = &H20000
Const CF_SCREENFONTS = &H1
Const CF_SCRIPTSONLY = &H400
Const CF_SELECTSCRIPT = &H400000
Const CF_SHOWHELP = &H4
Const CF_TTONLY = &H40000
Const CF_USESTYLE = &H80
Const CF_WYSIWYG = &H8000
Const BOLD_FONTTYPE = &H100
Const ITALIC_FONTTYPE = &H200
Const PRINTER_FONTTYPE = &H4000
Const REGULAR_FONTTYPE = &H400
Const SCREEN_FONTTYPE = &H2000
Const SIMULATED_FONTTYPE = &H8000

**Used By**

ChooseFont

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** August 19, 1999

This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/c/choosefont_type.html)

Go back to the Windows API Guide home page.

E-mail: [vbapi@vbapi.com](mailto:vbapi@vbapi.com) [Send Encrypted E-Mail](http://216.26.168.92/vbapi/ref/c/choosefont_type.html)
CURRENCYFMT Structure

Type CURRENCYFMT
    NumDigits As Long
    LeadingZero As Long
    Grouping As Long
    lpDecimalSep As String
    lpThousandSep As String
    NegativeOrder As Long
    PositiveOrder As Long
    lpCurrencySymbol As String
End Type

Description & Usage

The CURRENCYFMT structure stores information about how to format a currency value for display. This structure allows a program to specify how it wants currency to be displayed, overriding the format used by a locale.

Visual Basic-Specific Issues

None.

Data Members

NumDigits
    The number of digits to display after the decimal point.

LeadingZero
    If zero, do not pad the space to the right of the decimal point with zeros if there are fewer fractional digits than specified by NumDigits. If nonzero, then do pad the space. For example, if NumDigits is 3 and the number to display is $1.23, setting this data member to zero displays the number as "$1.23". Setting this data member to any other value displays "$1.230".

Grouping
The number of digits to include in each group to the left of the decimal point. Typically, groups of three are used (e.g., $1,234,567). Values in the range of 0-9 are valid.

*lpDecimalSep*

The character to use for the decimal point.

*lpThousandSep*

The character to use for the grouping separator.

**NegativeOrder**

One of the following values specifying how to represent a negative currency value. An example of each choice is shown in the following list for the value of negative $1.1.

0  
($)1.1)

1  
-$1.1

2  
$-1.1

3  
$1.1-

4  
(1.1$)

5  
-1.1$

6  
1.1-$

7  
1.1$-

8  
-1.1 $ (space before $)

9  
-$1.1 (space after $)

10  
1.1 $- (space before $)

11  
$ 1.1- (space after $)

12  
$ -1.1 (space after $)

13  
1.1- $ (space before $)

14  
($ 1.1) (space after $)

15  
(1.1 $) (space before $)

**PositiveOrder**

One of the following values specifying how to display a positive currency value. An example of
each choice is shown in the following list for the value of $1.1.

0
  $1.1
1
  1.1$
2
  $ 1.1 (space after $)
3
  1.1 $ (space before $)

\textit{lpCurrencySymbol}

The symbol used to represent the currency. For example, US dollars are typically represented by the $ character.

**Used By**

\texttt{GetCurrencyFormat}

---

**Last Modified:** April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. \textcolor{blue}{Copyright Information} \textbf{Revised October 29, 2000}
Go back to the Windows API Guide home page.
E-mail: \texttt{vbapi@vbapi.com} Send Encrypted E-Mail
This page is at \texttt{http://www.vbapi.com/ref/n/currencyfmt.html}
DEVMODE Structure

Type DEVMODE

    dmDeviceName As String * 32
    dmSpecVersion As Integer
    dmDriverVersion As Integer
    dmSize As Integer
    dmDriverExtra As Integer
    dmFields As Long
    dmOrientation As Integer
    dmPageSize As Integer
    dmPaperLength As Integer
    dmPaperWidth As Integer
    dmScale As Integer
    dmCopies As Integer
    dmDefaultSource As Integer
    dmPrintQuality As Integer
    dmColor As Integer
    dmDuplex As Integer
    dmYResolution As Integer
    dmTTOption As Integer
    dmCollate As Integer
    dmFormName As String * 32
    dmUnusedPadding As Integer
    dmBitsPerPixel As Integer
    dmPelsWidth As Long
    dmPelsHeight As Long
    dmDisplayFlags As Long
    dmDisplayFrequency As Long

' The following only appear in Windows 95, 98, 2000
    dmICMMethod As Long
    dmICMIntent As Long
    dmMediaType As Long
    dmDitherType As Long
    dmReserved1 As Long
    dmReserved2 As Long
The following only appear in Windows 2000

dmPanningWidth As Long
dmPanningHeight As Long

End Type

Description & Usage

The **DEVMODE** structure stores information about various settings and properties of a device, such as a printer. Some of the properties only apply to certain devices; for example, the `dmDisplayFrequency` has no relevant meaning for a printer. To determine which data members of the structure contain useful information, check the flags set in `dwFields`.

Visual Basic-Specific Information

None.

Data Members

- **dmDeviceName**
  The name of the device.

- **dmSpecVersion**
  The version number of the device's initialization information specification.

- **dmDriverVersion**
  The version number of the device driver.

- **dmSize**
  The size of the structure, in bytes.

- **dmDriverExtra**
  The number of bytes of information trailing the structure in memory.

- **dmFields**
  A combination of the following flags specifying which of the rest of the structure's members contain information about the device:
  - DM_ORIENTATION
    `dmOrientation` contains information.
  - DM_PAPERSIZE
    `dmPaperSize` contains information.
  - DM_PAPERLENGTH
    `dmPaperLength` contains information.
  - DM_PAPERWIDTH
    `dmPaperWidth` contains information.
  - DM_SCALE
Devmode contains information.

DM_Copies contains information.

DM_DefaultSource contains information.

DM_PrintQuality contains information.

DM_Color contains information.

DM_Duplex contains information.

DM_YResolution contains information.

DM_TTOption contains information.

DM_Collate contains information.

DM_FormName contains information.

DM_LogPixels contains information.

DM_BitsPerPixel contains information.

DM_PelsWidth contains information.

DM_PelsHeight contains information.

DM_DisplayFlags contains information.

DM_DisplayFrequency contains information.

DM_ICMMethod contains information.

Win 95/98 only: dmICMMethod contains information.

DM_ICMIntent contains information.

Windows 95, 98, 2000: dmICMIntent contains information.

DM_MediaType contains information.

Windows 95, 98, 2000: dmMediaType contains information.

DM_DitherType contains information.

Windows 95, 98, 2000: dmDitherType contains information.

DM_PanningWidth contains information.


DM_PanningHeight contains information.

Windows 2000: dmPanningHeight contains information.
**dmOrientation**

One of the following flags specifying the orientation of the printer paper:

- DMORIENT_PORTRAIT  
  Portrait (tall) mode.
- DMORIENT_LANDSCAPE  
  Landscape (wide) mode.

**dmPageSize**

If nonzero, one of the following flags specifying the size of the printer paper (or some other value specifying a paper size). If zero, the paper size is determined by *dmPaperLength* and *dmPaperWidth*.

- DMPAPER_LETTER  
  Letter, 8.5 x 11 inches.
- DMPAPER_LEGAL  
  Legal, 8.5 x 14 inches.
- DMPAPER_10X11  
  10 x 11 inches.
- DMPAPER_10X14  
  10 x 14 inches.
- DMPAPER_11X17  
  11 x 17 inches.
- DMPAPER_15X11  
  15 x 11 inches.
- DMPAPER_9X11  
  9 x 11 inches.
- DMPAPER_A_PLUS  
  A plus sheet.
- DMPAPER_A2  
  A2 sheet.
- DMPAPER_A3  
  A3 sheet, 297 x 420 millimeters.
- DMPAPER_A3_EXTRA  
  A3 extra sheet.
- DMPAPER_A3_EXTRA_TRANSVERSE  
  A3 extra transverse sheet.
- DMPAPER_A3_TRANSVERSE  
  A3 transverse sheet.
- DMPAPER_A4  
  A4 sheet, 210 x 297 millimeters.
- DMPAPER_A4_EXTRA  
  A4 extra sheet.
- DMPAPER_A4_PLUS  
  A4 plus sheet.
- DMPAPER_A4_TRANSVERSE  
  A4 transverse sheet.
A4 transverse sheet.

DMPAPER_A4SMALL
    A4 small sheet, 210 x 297 millimeters.

DMPAPER_A5
    A5 sheet, 148 x 210 millimeters.

DMPAPER_A5_EXTRA
    A5 extra sheet.

DMPAPER_A5_TRANSVERSE
    A5 transverse sheet.

DMPAPER_B_PLUS
    B plus sheet.

DMPAPER_B4
    B4 sheet, 250 x 354 millimeters.

DMPAPER_B5
    B5 sheet, 192 x 257 millimeters.

DMPAPER_B5_EXTRA
    B5 extra sheet.

DMPAPER_B5_TRANSVERSE
    B5 transverse sheet.

DMPAPER_CSHEET
    C sheet, 17 x 22 inches.

DMPAPER_DSHEET
    D sheet, 22 x 34 inches.

DMPAPER_ENV_10
    #10 envelope, 4.125 x 9.5 inches.

DMPAPER_ENV_11
    #11 envelope, 4.5 x 10.375 inches.

DMPAPER_ENV_12
    #12 envelope, 4.75 x 11 inches.

DMPAPER_ENV_14
    #14 envelope, 5 x 11.5 inches.

DMPAPER_ENV_9
    #9 envelope, 3.875 x 8.875 inches.

DMPAPER_ENV_B4
    B4 envelope, 250 x 353 millimeters.

DMPAPER_ENV_B5
    B5 envelope, 176 x 250 millimeters.

DMPAPER_ENV_B6
    B6 envelope, 176 x 125 millimeters.

DMPAPER_ENV_C3
    C3 envelope, 324 x 458 millimeters.

DMPAPER_ENV_C4
    C4 envelope, 229 x 324 millimeters.
DMPAPER_ENV_C5
  C5 envelope, 162 x 229 millimeters.
DMPAPER_ENV_C6
  C6 envelope, 114 x 162 millimeters.
DMPAPER_ENV_C65
  C65 envelope, 114 x 229 millimeters.
DMPAPER_ENV_DL
  DL envelope, 110 x 220 millimeters.
DMPAPER_ENV_INVITE
  Invitation envelope.
DMPAPER_ENV_ITALY
  Italy envelope, 110 x 230 millimeters.
DMPAPER_ENV_MONARCH
  Monarch envelope, 3.875 x 7.5 inches.
DMPAPER_ENV_PERSONAL
  Personal (6.75) envelope, 3.625 x 6.5 inches.
DMPAPER_ESHEET
  E sheet, 34 x 44 inches.
DMPAPER_EXECUTIVE
  Executive, 7.25 x 10.5 inches.
DMPAPER_FANFOLD_LGL_GERMAN
  German legal fanfold, 8.5 x 13 inches.
DMPAPER_FANFOLD_STD_GERMAN
  German standard fanfold, 8.5 x 12 inches.
DMPAPER_FANFOLD_US
  US standard fanfold, 14.875 x 11 inches.
DMPAPER_FIRST
  Same as DMPAPER_LETTER.
DMPAPER_FOLIO
  Folio, 8.5 x 13 inches.
DMPAPER_ISO_B4
  ISO B4 sheet.
DMPAPER_JAPANESE_POSTCARD
  Japanese postcard.
DMPAPER_LAST
  Same as DMPAPER_FANFOLD_LGL_GERMAN.
DMPAPER_LEDGER
  Ledger, 17 x 11 inches.
DMPAPER_LEGAL_EXTRA
  Legal extra.
DMPAPER_LETTER_EXTRA
  Letter extra.
DMPAPER_LETTER_EXTRA_TRANSVERSE
Letter extra transverse.

DMPAPER_LETTER_PLUS
Letter plus.

DMPAPER_LETTER_TRANSVERSE
Letter transverse.

DMPAPER_LETTERSMALL
Letter small, 8.5 x 11 inches.

DMPAPER_NOTE
Note, 8.5 x 11 inches.

DMPAPER_QUARTO
Quarto, 215 x 275 millimeters.

DMPAPER_STATEMENT
Statement, 5.5 x 8.5 inches.

DMPAPER_TABLOID
Tabloid, 11 x 17 inches.

DMPAPER_TABLOID_EXTRA
Tabloid extra.

DMPAPER_USER
User-defined size.

dmPaperLength
The length of the printer paper, measured in tenths of a millimeter.

dmPaperWidth
The width of the printer paper, measured in tenths of a millimeter.

dmScale
The scale percentage factor (e.g., 100 means 100%, or no, scaling; 200 means two times the size, etc.).

dmCopies
The number of document copies to print, if the device supports it.

dmDefaultSource
One of the following flags specifying the printer's source of paper:

DMBIN_ONLYONE
There is only one paper source.

DMBIN_UPPER
Upper bin.

DMBIN_LOWER
Lower bin.

DMBIN_MIDDLE
Middle bin.

DMBIN_MANUAL
Manual loading.

DMBIN_ENVELOPE
Envelope bin.

DMBIN_ENVMANUAL
Manual-loading envelope.

DMBIN_AUTO
Automatic loading.

DMBIN_TRACTOR
Tractor loading.

DMBIN_SMALLFMT
Small format loading.

DMBIN_LARGEFMT
Large format loading.

DMBIN_LARGECAPACITY
Large-capacity bin.

DMBIN_CASSETTE
Cassette.

DMBIN_FORMSOURCE
Form paper source.

*dmPrintQuality*
Either one of the following flags specifying the printer's print quality setting, or a positive value specifying the printer's dots per inch (DPI) rating.

DMRES_DRAFT
Draft-quality output.

DMRES_LOW
Low-quality output.

DMRES_MEDIUM
Medium-quality output.

DMRES_HIGH
High-quality output.

*dmColor*
One of the following flags specifying whether the device supports color:

DMCOLOR_MONOCHROME
The device does not support color output.

DMCOLOR_COLOR
The device supports color output.

*dmDuplex*
One of the following flags specifying the printer's double-sided (duplex) printing capability:

DMDUP_SIMPLEX
Configured for single-sided printing.

DMDUP_VERTICAL
Configured for double-sided printing with vertical page turning.

DMDUP_HORIZONTAL
Configured for double-sided printing with horizontal page turning.

*dmYResolution*
The number of the vertical dots per inch of the printer. If this value contains useful data, the number of horizontal dots per inch is inside *dmPrintQuality*.
**dmTTOption**

One of the following flags specifying how the printer prints TrueType fonts:

**DMTT_BITMAP**

The printer prints TrueType fonts as graphics (default for dot-matrix printers).

**DMTT_DOWNLOAD**

The printer downloads TrueType fonts as soft fonts (default for Hewlett-Packard printers using Printer Control Language).

**DMTT_SUBDEV**

The printer substitutes device fonts for TrueType fonts (default for PostScript printers).

**dmUnusedPadding**

Reserved -- set to 0. This member merely takes up space to align other members in memory.

**dmCollate**

One of the following flags specifying whether the printer can collate copies:

**DMCOLLATE_FALSE**

Does not collate pages when printing multiple copies.

**DMCOLLATE_TRUE**

Does collate pages when printing multiple copies.

**dmFormName**

**Windows NT, 2000:** The name of the type of paper loaded in the printer.

**dmBitsPerPel**

The number of color bits used per pixel on the display device.

**dmPelsWidth**

The width of the display, measured in pixels.

**dmPelsHeight**

The height of the display, measured in pixels.

**dmDisplayFlags**

A combination of the following flags specifying the device's display mode:

**DM_GRAYSCALE**

The display does not support color. (If this flag is omitted, assume color is supported.)

**DM_INTERLACED**

The display is interlaced.

**dmDisplayFrequency**

The display frequency of the display, measured in Hz.

**dmICMMethod**

**Windows 95, 98, 2000:** Either one of the following flags specifying how image color matching (ICM) is supported, or a device-defined value greater than 256:

**DMICMMETHOD_NONE**

ICM is disabled.

**DMICMMETHOD_SYSTEM**

ICM is handled by Windows.

**DMICMMETHOD_DRIVER**

ICM is handled by the device driver.
ICM is handled by the device.

**dmICMIntent**

**Windows 95, 98, 2000:** Either one of the following flags specifying the image color matching (ICM) method used when ICM is not intrinsically supported, or a device-defined value greater than 256:

- DMICM_SATURATE
  - Color matching attempts to optimize color saturation.
- DMICM_CONTRAST
  - Color matching attempts to optimize color contrast.
- DMICM_COLORMETRIC
  - Color matching attempts to match the exact color requested.

**dmMediaType**

**Windows 95, 98, 2000:** Either one of the following flags specifying what type of medium the printer is printing on, or a device-defined value greater than 256:

- DMMEDIA_STANDARD
  - Plain paper.
- DMMEDIA_GLOSSY
  - Glossy paper.
- DMMEDIA_TRANSPARENCY
  - Transparent film.

**dmDitherType**

**Windows 95, 98, 2000:** Either one of the following flags specifying the dithering method used by the device, or a device-defined value greater than 256:

- DMDITHER_NONE
  - No dithering.
- DMDITHER_COARSE
  - Dithering with a coarse brush.
- DMDITHER_FINE
  - Dithering with a fine brush.
- DMDITHER_LINEART
  - Line art dithering, which makes well-defined borders between black, white, and gray.
- DMDITHER_GRAYSCALE
  - Grayscaling.

**dmReserved1**

**Windows 95, 98, 2000:** Reserved -- set to 0.

**dmReserved2**

**Windows 95, 98, 2000:** Reserved -- set to 0.

**dmPanningWidth**

**Windows 2000:** Reserved -- set to 0.

**dmPanningHeight**

**Windows 2000:** Reserved -- set to 0.
Constant Definitions

Const DM_ORIENTATION = &H1
Const DM_PAPERSIZE = &H2
Const DM_PAPERLENGTH = &H4
Const DM_PAPERWIDTH = &H8
Const DM_SCALE = &H10
Const DM_COPIES = &H100
Const DM_DEFAULTSOURCE = &H200
Const DM_PRINTQUALITY = &H400
Const DM_COLOR = &H800
Const DM_DUPLEX = &H1000
Const DM_YRESOLUTION = &H2000
Const DM_TTOPTION = &H4000
Const DM_COLLATE = &H8000
Const DM_FORMNAME = &H10000
Const DM_LOGPIXELS = &H20000
Const DM_BITSPERPEL = &H40000
Const DM_PELSWIDTH = &H80000
Const DM_PELSHEIGHT = &H100000
Const DM_DISPLAYFLAGS = &H200000
Const DM_DISPLAYFREQUENCY = &H400000
Const DM_ICMMETHOD = &H800000
Const DM_ICMINTENT = &H1000000
Const DM_MEDIATYPE = &H2000000
Const DM_DITHERTYPE = &H4000000
Const DM_PANNINGWIDTH = &H20000000
Const DM_PANNINGHEIGHT = &H40000000
Const DMORIENT_PORTRAIT = 1
Const DMORIENT_LANDSCAPE = 2
Const DMPAPER_LETTER = 1
Const DMPAPER_LEGAL = 5
Const DMPAPER_10X11 = 45
Const DMPAPER_10X14 = 16
Const DMPAPER_11X17 = 17
Const DMPAPER_15X11 = 46
Const DMPAPER_9X11 = 44
Const DMPAPER_A_PLUS = 57
Const DMPAPER_A2 = 66
Const DMPAPER_A3 = 8
Const DMPAPER_A3_EXTRA = 63
Const DMPAPER_A3_EXTRA_TRANSVERSE = 68
Const DMPAPER_A3_TRANSVERSE = 67
Const DMPAPER_A4 = 9
Const DMPAPER_A4_EXTRA = 53
Const DMPAPER_A4_PLUS = 60
Const DMPAPER_A4_TRANSVERSE = 55
Const DMPAPER_A4SMALL = 10
Const DMPAPER_A5 = 11
Const DMPAPER_A5_EXTRA = 64
Const DMPAPER_A5_TRANSVERSE = 61
Const DMPAPER_B_PLUS = 58
Const DMPAPER_B4 = 12
Const DMPAPER_B5 = 13
Const DMPAPER_B5_EXTRA = 65
Const DMPAPER_B5_TRANSVERSE = 62
Const DMPAPER_CSHEET = 24
Const DMPAPER_DSHEET = 25
Const DMPAPER_ENV_10 = 20
Const DMPAPER_ENV_11 = 21
Const DMPAPER_ENV_12 = 22
Const DMPAPER_ENV_14 = 23
Const DMPAPER_ENV_9 = 19
Const DMPAPER_ENV_B4 = 33
Const DMPAPER_ENV_B5 = 34
Const DMPAPER_ENV_B6 = 35
Const DMPAPER_ENV_C3 = 29
Const DMPAPER_ENV_C4 = 30
Const DMPAPER_ENV_C5 = 28
Const DMPAPER_ENV_C6 = 31
Const DMPAPER_ENV_C65 = 32
Const DMPAPER_ENV_DL = 27
Const DMPAPER_ENV_INVITE = 47
Const DMPAPER_ENV_ITALY = 36
Const DMPAPER_ENV_MONARCH = 37
Const DMPAPER_ENV_PERSONAL = 38
Const DMPAPER_ESHEET = 26
Const DMPAPER_EXECUTIVE = 7
Const DMPAPER_FANFOLD_LGL_GERMAN = 41
Const DMPAPER_FANFOLD_STD_GERMAN = 40
Const DMPAPER_FANFOLD_US = 39
Const DMPAPER_FIRST = 1
Const DMPAPER_FOLIO = 14
Const DMPAPER_ISO_B4 = 42
Const DMPAPER_JAPANESE_POSTCARD = 43
Const DMPAPER_LAST = 41
Const DMPAPER_LEDGER = 4
Const DMPAPER_LEGAL_EXTRA = 51
Const DMPAPER_LETTER_EXTRA = 50
Const DMPAPER_LETTER_EXTRA_TRANSVERSE = 56
Const DMPAPER_LETTER_PLUS = 59
Const DMPAPER_LETTER_TRANSVERSE = 54
Const DMPAPER_LETTERSMALL = 2
Const DMPAPER_NOTE = 18
Const DMPAPER_QUARTO = 15
Const DMPAPER_STATEMENT = 6
Const DMPAPER_TABLOID = 3
Const DMPAPER_TABLOID_EXTRA = 52
Const DMPAPER_USER = 256
Const DMBIN_ONLYONE = 1
Const DMBIN_UPPER = 1
Const DMBIN_LOWER = 2
Const DMBIN_MIDDLE = 3
Const DMBIN_MANUAL = 4
Const DMBIN_ENVELOPE = 5
Const DMBIN_ENVMANUAL = 6
Const DMBIN_AUTO = 7
Const DMBIN_TRACTOR = 8
Const DMBIN_SMALLFMT = 9
Const DMBIN_LARGEFMT = 10
Const DMBIN_LARGECAPACITY = 11
Const DMBIN_CASSETTE = 14
Const DMBIN_FORMSOURCE = 15
Const DMRES_DRAFT = -1
Const DMRES_LOW = -2
Const DMRES_MEDIUM = -3
Const DMRES_HIGH = -4
Const DMCOLOR_MONOCHROME = 1
Const DMCOLOR_COLOR = 2
Const DMDUP_SIMPLEX = 1
Const DMDUP_VERTICAL = 2
Const DMDUP_HORIZONTAL = 3
Const DMTT_BITMAP = 1
Const DMTT_DOWNLOAD = 2
Const DMTT_SUBDEV = 4
Const DMCOLLATE_FALSE = 0
Const DMCOLLATE_TRUE = 1
Const DM_GRAYSCALE = 1
Const DM_INTERLACED = 2
Const DMICMMETHOD_NONE = 1
Const DMICMMETHOD_SYSTEM = 2
Const DMICMMETHOD_DRIVER = 3
Const DMICMMETHOD_DEVICE = 4
Const DMICM_SATURATE = 1
Const DMICM_CONTRAST = 2
Const DMICM_COLORMETRIC = 3
Const DMMEDIA_STANDARD = 1
Const DMMEDIA_GLOSSY = 2
Const DMMEDIA_TRANSPARENCY = 3
Const DMDITHER_NONE = 1
Const DMDITHER_COARSE = 2
Const DMDITHER_FINE = 3
Const DMDITHER_LINEART = 4
Const DMDITHER_GRAYSCALE = 5

**Used By**

ChangeDisplaySettings, CreateDC, EnumDisplaySettings, JOB_INFO_2, PRINTDLG_TYPE, PRINTER_DEFAULTS, PRINTER_INFO_2

Go back to the Structure listing.
Go back to the Reference section index.

---

**Last Modified:** January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/d/devmode.html](http://www.vbapi.com/ref/d/devmode.html)
DEVNAMES Structure

Type DEVNAMES
    wDriverOffset As Integer
    wDeviceOffset As Integer
    wOutputOffset As Integer
    wDefault As Integer
    extra As String * 100
End Type

DEVNAMES-type variables store some information about a device. This information includes the device driver name, the device name, and the names of any output ports it uses. Note that instead of storing strings in the usual way, this structure puts all three strings into extra, where null characters separate them. The offset values specify the location of these strings in extra, measured in bytes from the beginning of the structure. For example, the very first character in extra would have an offset of 8. See the example for the PrintDlg function for a demonstration of using this structure.

wDriverOffset
    The offset of the string in extra identifying the name of the device driver filename (without the extension).

wDeviceOffset
    The offset of the string in extra identifying the name of the device.

wOutputOffset
    The offset of the string in extra identifying the output port(s) which the device uses, separated by commas.

wDefault
    If non-zero, the information in the structure identifies the default device of its type. If zero, the information does not necessarily describe the default device.

extra
    Buffer which holds the three strings identified by wDriverOffset, wDeviceOffset, and wOutputOffset.

Used by: PRINTDLG_TYPE

Go back to the alphabetical Structure listing.
DOCINFO Structure

Type DOCINFO
    cbSize As Long
    lpszDocName As String
    lpszOutput As Long
    lpszDatatype As String
    fwType As Long
End Type

Description & Usage

The DOCINFO structure holds information about a print job. This information is mainly used to give a description of the document to the print spooler. It also specifies whether to send the data to the printer or to a print file, as well as specifying a few additional options.

Visual Basic-Specific Issues

None.

Data Members

- **cbSize**
  - The size in bytes of the structure.
- **lpszDocName**
  - The name of the document being printed.
- **lpszOutput**
  - To send the output to a print file, this is a pointer to a null-terminated string identifying the file to create. To send the output to the printer itself, set this to 0.
- **lpszDatatype**
  - The data type of the document being printed, if applicable.
- **fwType**
  - Zero or more of the following flags specifying additional options for printing:
DI_APPBANDING
If the program uses banding, set this flag to provide optimal performance during printing.

DI_ROPS_READ_DESTINATION
Use raster operations which involve reading the destination surface.

Constant Definitions

Const DI_APPBANDING = &H1
Const DI_ROPS_READ_DESTINATION = &H2

Used By

StartDoc

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: November 2, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/d/docinfo.html
ENUMLOGFONT Structure

Type ENUMLOGFONT
    elfLogFont As LOGFONT
    elfFullName As String * 64
    elfStyle As String * 64
End Type

Description & Usage

The ENUMLOGFONT structure holds information about a logical font which has been enumerated. The structure not only identifies the regular logical font information but also some additional information as well. Windows 95, 98: If the font is not a TrueType font, the elfLogFont member must be used to get all of the font’s information. The other data members of this structure will not contain any useful data.

Visual Basic-Specific Issues

None.

Data Members

elfLogFont
    Logical font information about the enumerated font.

elfFullName
    A null-terminated string identifying the full name of the enumerated font. Windows 95, 98: If the font is not a TrueType font, this string does not contain any useful information.

elfStyle
    A null-terminated string identifying the style attributes of the enumerated font. Windows 95, 98: If the font is not a TrueType font, this string does not contain any useful information.

Used By
ENUMLOGFONTTEX Structure

Type ENUMLOGFONTTEX
    elfLogFont As LOGFONT
    elfFullName As String * 64
    elfStyle As String * 32
    elfScript As String * 32
End Type

Description & Usage

The ENUMLOGFONTTEX structure holds information about a logical font which has been enumerated. The structure not only identifies the regular logical font information but also some additional information as well. Windows 95, 98: If the font is not a TrueType font, the elfLogFont member must be used to get all of the font's information. The other data members of this structure will not contain any useful data.

Visual Basic-Specific Issues

None.

Data Members

elfLogFont
Logical font information about the enumerated font.

elfFullName
A null-terminated string identifying the full name of the enumerated font. Windows 95, 98: If the font is not a TrueType font, this string does not contain any useful information.

elfStyle
A null-terminated string identifying the style attributes of the enumerated font. Windows 95, 98: If the font is not a TrueType font, this string does not contain any useful information.

elfScript
A null-terminated string identifying the character set of the enumerated font. Windows 95, 98: If the font is not a TrueType font, this string does not contain any useful information.
Used By

EnumFontFamExProc

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
FILETIME Structure

Type FILETIME
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Description & Usage

The FILETIME structure holds a date and time associated with a file. The structure identifies a 64-bit integer specifying the number of 100-nanosecond intervals which have passed since January 1, 1601. This 64-bit value is split into the two dwords stored in the structure.

Visual Basic-Specific Issues

None.

Parameters

**dwLowDateTime**
The low-order dword of the 64-bit integer specifying the date and time.

**dwHighDateTime**
The high-order dword of the 64-bit integer specifying the date and time.

Used By

BY_HANDLE_FILE_INFORMATION, CompareFileTime, FileTimeToLocalFileTime, FileTimeToSystemTime, GetFileTime, GetSystemTimeAsFileTime, LocalFileTimeToFileTime, RegEnumKeyEx, SetFileTime, SystemTimeToFileTime, WIN32_FIND_DATA

Go back to the alphabetical Structure listing.
FILTERKEYS Structure

Type FILTERKEYS
    cbSize As Long
    dwFlags As Long
    iWaitMSec As Long
    iDelayMSec As Long
    iRepeatMSec As Long
    iBounceMSec As Long
End Type

FILTERKEYS-type variables store information relating to the FilterKeys accessibility feature. FilterKeys helps filter out unwanted and accidental keypresses by controlling settings such as the repeat rate and requiring keys to be held for a certain time before they are accepted. This structure stores the settings of FilterKeys.

`cbSize`

The size in bytes of the structure.

`dwFlags`

Zero or more of the following flags specifying various FilterKeys settings and properties:

FKF_AVAILABLE = &H2
    The FilterKeys accessibility feature is available.
FKF_CLICKON = &H40
    Play a clicking sound whenever a key is pressed or accepted.
FKF_CONFIRMHOTKEY = &H8
    Win 95/98 only: Display a confirmation dialog whenever the user activates FilterKeys via the hot key.
FKF_FILTERKEYSON = &H1
    FilterKeys is currently on.
FKF_HOTKEYACTIVE = &H4
    The user can toggle FilterKeys by using the hot key: holding the right Shift key for eight seconds.
FKF_HOTKEYSOUND = &H10
    Play a sound when the user toggles FilterKeys via the hot key.
FKF_INDICATOR = &H20
    Win 95/98 only: Display an icon in the tray while FilterKeys in on.
**iWaitMSec**
The length of time, in milliseconds, the user must hold down a key before it is accepted.

**iDelayMSec**
The length of time, in milliseconds, the user must hold down a key before it begins to repeat.

**iRepeatMSec**
The length of time, in milliseconds, between each repetition of a key.

**iBounceMSec**
The length of time, in milliseconds, before subsequent presses of the same key will be accepted.

**Used by:** [SystemParametersInfo](http://www.vbapi.com/ref/f/systemparametersinfo.html)

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/copyright.html).
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/f/filterkeys.html](http://www.vbapi.com/ref/f/filterkeys.html)
FONTSIGNATURE Structure

Type FONTSIGNATURE
   fsUsb(0 To 3) As Long
   fsCsb(0 To 1) As Long
End Type

Description & Usage

The FONTSIGNATURE structure holds the font signature of a font. The font signature is composed of a 128-bit Unicode subset bitfield (USB) and a 64-bit code-page bitfield (CPB). Both of these values are split into multi-element arrays of 32-bit integers. For each array, the lowest-order dword appears as the first array element, and the highest-order dword appears as the last array element.

Visual Basic-Specific Issues

None.

Data Members

*fsUsb*

A Unicode subset bitfield (USB), a 128-bit value, identifying up to 126 Unicode subranges. Each bit except the two most significant ones identify one of 126 possibly Unicode subranges, according to the ISO 10646 standard.

*fsCsb*

A code-page bitfield (CPB), a 64-bit value, identifying a specific character set or code page. Windows code pages are in the low-order dword; non-Windows code pages are in the high-order dword.

Used By

NEWTEXTMETRICEX
Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: October 27, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/f/fontsignature.html
HARDWAREINPUT Structure

Type HARDWAREINPUT
  uMsg As Long
  wParamL As Integer
  wParamH As Integer
End Type

Description & Usage

The HARDWAREINPUT structure holds information about a message synthesized by some generic (non-keyboard, non-mouse) input hardware. The data members of the structure specify the information associated with the message generated.

Visual Basic-Specific Issues

None.

Data Members

uMsg
  The message identifier of the message generated by the input hardware.

wParamL
  The low-order word of the message’s first parameter.

wParamH
  The high-order word of the message’s first parameter.

Used By

INPUT_TYPE

Go back to the alphabetical Structure listing.
HELPINFO Structure

Type HELPINFO
    cbSize As Long
    iContextType As Long
    iCtrlId As Long
    hItemHandle As Long
    dwContextId As Long
    MousePos As POINT_TYPE
End Type

Description & Usage

The HELPINFO structure holds information about a help request. The structure holds details about the object or other item about which help information is requested by the user. The structure does not specify the help documentation itself, but rather the object for which context-sensitive help is wanted.

Visual Basic-Specific Issues

None.

Data Members

cbSize
    The size in bytes of the structure.

iContextType
    One of the following flags specifying the type of context for which help information is requested:
    HELPINFO_MENUITEM
        Help information is requested for a menu item.
    HELPINFO_WINDOW
        Help information is requested for a control or window.

iCtrlId
    An identifier of the window, control, or menu item for which context-sensitive help is desired.
**hItemHandle**
A handle to the window or control (if iContextType is HELPINFO_WINDOW) or an identifier of the associated menu (if iContextType is HELPINFO_MENUITEM) for which context-sensitive help is desired.

**dwContextId**
The help context identifier of the window or control.

**MousePos**
The coordinates of the mouse cursor when help information was requested.

## Constant Definitions

```
Const HELPINFO_MENUITEM = 2
Const HELPINFO_WINDOW = 1
```

## Used By

- MsgBoxCallback
- WM_HELP

Back to the Structure list.
Back to the Reference section.

---

**Last Modified:** January 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](#) **Revised October 29, 2000**

Go back to the [Windows API Guide home page.](#)
E-mail: vbapi@vbapi.com [Send Encrypted E-Mail]
This page is at [http://www.vbapi.com/ref/h/helpinfo.html](http://www.vbapi.com/ref/h/helpinfo.html)
HELPWININFO Structure

Type HELPWININFO
    wStructSize As Long
    x As Long
    y As Long
    dx As Long
    dy As Long
    wMax As Long
    rgchMember As String * 2
End Type

Description & Usage

The HELPWININFO structure holds information about the size and position of a primary or secondary Windows Help window. This information is used to reposition such a window. "Normal" screen coordinates are not used by this structure! Instead, the screen is divided into 1024 units both horizontally and vertically (regardless of the actual resolution). This coordinate system is used by the data members which specify coordinates.

Visual Basic-Specific Issues

None.

Data Members

wStructSize
    The size in bytes of the structure.

x
    The x-coordinate of the upper-left corner of the window.

y
    The y-coordinate of the upper-left corner of the window.

dx
The width of the window.

dy
The height of the window.

wMax
Exactly one of the following flags specifying how to show the window:

SW_HIDE
  Hide the window and make a different window the active window.

SW_MINIMIZE
  Minimize the window and activate the top window in the \textit{z-order}.

SW_RESTORE
  Same as SW_SHOWNORMAL.

SW_SHOW
  Activate the window and display it in its current size and position.

SW_SHOWMAXIMIZED
  Activate and maximize the window.

SW_SHOWMINIMIZED
  Activate and minimize the window.

SW_SHOWMINNOACTIVE
  Minimize the window, leaving whichever window was previously the active window active.

SW_SHOWNA
  Display the window in its current size and position, leaving whichever window was previously the active window active.

SW_SHOWNOACTIVATE
  Display the window in its most recent size and position, leaving whichever window was previously the active window active.

SW_SHOWNORMAL
  Activate and display the window, restoring it to its original size and position.

rgchMember
The name of the window to set the size and position of.

\textbf{Constant Definitions}

\begin{verbatim}
Const SW_HIDE = 0
Const SW_MINIMIZE = 6
Const SW_RESTORE = 9
Const SW_SHOW = 5
Const SW_SHOWMAXIMIZED = 3
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMINNOACTIVE = 7
Const SW_SHOWNA = 8
Const SW_SHOWNOACTIVATE = 4
\end{verbatim}
Const SW_SHOWNORMAL = 1

Used By

WinHelp

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

Last Modified: August 13, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/h/helpwininfo.html
HIGHCONTRAST Structure

Type HIGHCONTRAST
    cbSize As Long
    dwFlags As Long
    lpszDefaultScheme As String
End Type

HIGHCONTRAST-type variables store information about the HighContrast accessibility feature. HighContrast aids users with poor vision by utilizing a high-contrast Windows color scheme to maximize visibility. The structure stores the settings for the HighContrast accessibility feature.

cbSize

The size in bytes of the structure.

dwFlags

Zero or more of the following flags specifying various settings and properties of HighContrast:
HCF_AVAILABLE = &H2
    The HighContrast accessibility feature is available.
HCF_CONFIRMHOTKEY = &H8
    Open a confirmation dialog box when the user activates HighContrast via the hot key.
HCF_HIGHCONTRASTON = &H1
    HighContrast is currently on.
HCF_HOTKEYACTIVE = &H4
    Allow the user to toggle HighContrast using the hot key: pressing Left Alt, Left Shift, and Print Screen simultaneously.
HCF_HOTKEYAVAILABLE = &H40
    The hot key can be enabled (this flag cannot be changed).
HCF_HOTKEYSOUND = &H10
    Play a sound when the user toggles HighContrast via the hot key.
HCF_INDICATOR = &H20
    Display an icon in the system tray while HighContrast is on.

lpszDefaultScheme

The name of the color scheme for HighContrast to use.

Used by: SystemParametersInfo
Go back to the alphabetical Structure listing.
Go back to the Reference section index.

This page is copyright © 2000 Paul Kuliniewicz. Copyright Information.
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/h/highcontrast.html
HOSTENT Structure

Type HOSTENT

    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long

End Type

Description & Usage

The HOSTENT structure stores information about a host computer on a network. Most of the information in the structure is stored as a set of pointers to the actual data.

Visual Basic-Specific Issues

To access the data pointed to by the structure, it is necessary to use functions such as CopyMemory, lstrcpy, and lstrlen to copy the data into the appropriate variables. See the example for gethostbyname for a demonstration of how this is done.

Data Members

h_name

A pointer to a null-terminated string specifying the fully-qualified domain name of the host computer.

h_aliases

A pointer to a null-terminated array of alternate domain names for the host computer.

h_addrtype

The address family of the base protocol used by the network.

h_length

The length in bytes of each address pointed to by h_addr_list.

h_addr_list
A pointer to a null-terminated list of addresses for the host computer. The addresses are returned in network byte order.

**Used By**

gethostbyaddr, gethostbyname

[Back to the Structure list.](#)
[Back to the Reference section.](#)
ICONMETRICS Structure

Type ICONMETRICS
   cbSize As Long
   iHorzSpacing As Long
   iVertSpacing As Long
   iTileWrap As Long
   lfFont As LOGFONT
End Type

ICONMETRICS-type variables hold information about various icon metrics. The icon metrics specify information about icons in Windows, such as the font used for their titles and the spacing used to display them. The structure stores these metrics.

   cbSize  
   The size in bytes of the structure.

   iHorzSpacing  
   The horizontal space, in pixels, allotted to each arranged icon.

   iVertSpacing  
   The vertical space, in pixels, allotted to each arranged icon.

   iTileWrap  
   Specifies whether the titles of icons are word-wrapped or not. Zero means the titles are not wrapped; a non-zero value means the titles are wrapped.

   lfFont  
   Information about the logical font used to display the icon titles.

Used by: SystemParametersInfo
**INITCOMMONCONTROLSEX_TYPE Structure**

```
Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
```

**Description & Usage**

The `INITCOMMONCONTROLSEX_TYPE` structure tells `InitCommonControlsEx` which common control classes to register. The structure holds a combination of flags specifying which classes are wanted.

**Visual Basic-Specific Issues**

Officially, the name of this structure is `INITCOMMONCONTROLSEX` (all capitals). However, since Visual Basic is case-insensitive, that name conflicts with the `InitCommonControlsEx` API function. Therefore, it is necessary to give it a different name -- in this case, by appending _TYPE to the end.

**Data members.**

*dwSize*

The length in bytes of the structure.

*dwICC*

A combination of the following flags specifying which common control classes to register:

- `ICC_ANIMATE_CLASS`
  The Animate control class.

- `ICC_BAR_CLASSES`
  The Toolbar, Status Bar, Trackbar, and Tooltip control classes.

- `ICC_COOL_CLASSES`
  The Rebar control class.
ICC_DATE_CLASSES
The Date and Time Picker control class.

ICC_HOTKEY_CLASS
The Hot Key control class.

ICC_INTERNET_CLASSES
The IP Address control class.

ICC_LISTVIEW_CLASSES
The List View and Header control classes.

ICC_PAGESCROLLER_CLASS
The Pager control class.

ICC_PROGRESS_CLASS
The Progress Bar control class.

ICC_TAB_CLASSES
The Tab and Tooltip control classes.

ICC_TREEVIEW_CLASSES
The Tree View and Tooltip control classes.

ICC_UPDOWN_CLASS
The Up-Down control class.

ICC_USEREX_CLASSES
The ComboBoxEx control class.

ICC_WIN95_CLASSES
The Animate, Header, Hot Key, List View, Progress Bar, Status Bar, Tab, Tooltip, Toolbar, Trackbar, Tree View, and Up-Down control classes. (In other words, almost all of them.)

Constant Definitions

Const ICC_ANIMATE_CLASS = &H80
Const ICC_BAR_CLASSES = &H4
Const ICC_COOL_CLASSES = &H400
Const ICC_DATE_CLASSES = &H100
Const ICC_HOTKEY_CLASS = &H40
Const ICC_INTERNET_CLASSES = &H800
Const ICC_LISTVIEW_CLASSES = &H1
Const ICC_PAGESCROLLER_CLASS = &H1000
Const ICC_PROGRESS_CLASS = &H20
Const ICC_TAB_CLASSES = &H8
Const ICC_TREEVIEW_CLASSES = &H2
Const ICC_UPDOWN_CLASS = &H10
Const ICC_USEREX_CLASSES = &H200
Const ICC_WIN95_CLASSES = &HFF
Used By

InitCommonControlsEx

Back to the Structure list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/initcommoncontrolsex.html
INPUT_TYPE Structure

Type INPUT_TYPE
    dwType As Long
    xi(0 To 23) As Byte
End Type

Description & Usage

The INPUT_TYPE structure holds information about an input event to be placed in the input stream. The input can be from the keyboard, the mouse, or some other hardware. Essentially, this structure merely identifies the source of an input event.

Visual Basic-Specific Issues

Officially, this structure is called INPUT. However, that violates the case-sensitive name spacing of Visual Basic because Visual Basic contains an intrinsic Input command. The Windows API Guide calls this structure INPUT_TYPE to avoid the naming collision.

Also, this structure does not officially have a data member called xi. That space in the structure is actually a "union" of three members: mi, ki, and hi, all of which occupy the same physical space in the structure and of which only one can be used at any one time. Each of those actual members are the contents of one of the three structures discussed below physically embedded in the structure. Because, unlike C++, Visual Basic does not have any analogous construct, a workaround must be reached. Therefore, as far as Visual Basic is concerned, a byte array called xi occupies that space, into which the contents of one of the possible structures must be copied. See the example for SendInput for a demonstration.

Data Members

dwType
    Exactly one of the following flags specifying which type of input event this structure contains:
    INPUT_MOUSE
The structure identifies a mouse input event.
INPUT_MOUSE
The structure identifies a keyboard input event.
INPUT_KEYBOARD
The structure identifies some other hardware input event.
INPUT_HARDWARE
Windows 98: The structure identifies some other hardware input event.

Buffer which holds the contents of a data structure, depending on the value of dwType. A MOUSEINPUT structure is used for mouse events, a KEYBDINPUT structure for keyboard events, and a HARDWAREINPUT structure for other hardware events. (See the SendInput example for a demonstration of using this parameter.)

Constant Definitions

Const INPUT_MOUSE = 0
Const INPUT_KEYBOARD = 1
Const INPUT_HARDWARE = 2

Used By

SendInput

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: October 9, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/input_type.html
ITEMIDLIST Structure

Type ITEMIDLIST
  mkid As SHITEMID
End Type

Description & Usage

The ITEMIDLIST structure stores an item identifier which identifies something. Usually, API functions merely use a pointer to this structure (often called a PIDL) instead of using the structure directly. This is because only API function actually use the contents of the structure; regular programs don't need to use it. This is also because the actual size of the structure (the size of mkid) can vary widely, so API functions often create the structure automatically and return a pointer to it, eliminating the need for the program to actually process its contents in any way.

Visual Basic-Specific Issues

None.

Data Members

mkid
  Holds the item identifier data.

Used By

BrowseCallbackProc, BROWSEINFO, SHAddToRecentDocs, SHBrowseForFolder, SHELLEXECUTEINFO, SHGetFileInfo, SHGetFolderLocation, SHGetPathFromIDList, SHGetSpecialFolderLocation

Go back to the alphabetical Structure listing.
JOB_INFO_1 Structure

Type JOB_INFO_1
    JobId As Long
    pPrinterName As String
    pMachineName As String
    pUserName As String
    pDocument As String
    pDatatype As String
    pStatus As String
    Status As Long
    Priority As Long
    Position As Long
    TotalPages As Long
    PagesPrinted As Long
    Submitted As SYSTEMTIME

End Type

Description & Usage

The JOB_INFO_1 structure holds information about a print job. The structure contains data which both identifies the originator of the print job and its current status in the print spooler.

Visual Basic-Specific Issues

None.

Data Members

JobId
    The unique numeric identifier of the print job.

pPrinterName
    The name of the printer for which the print job is spooled.
pMachineName
The name of the computer which created the print job.

pUserName
The name of the user who owns the print job.

pDocument
The name of the print job (usually the document being printed).

pDatatype
The type of data used to record the print job.

pStatus
The current status of the print job. If this string is empty, the print status can instead be found in Status.

Status
If necessary, a combination of the following flags identifying the current status of the print job:

- JOB_STATUS_BLOCKED_DEVQ
  The driver cannot print the print job.
- JOB_STATUS_DELETED
  The print job has been deleted.
- JOB_STATUS_DELETING
  The print job is being deleted.
- JOB_STATUS_ERROR
  An error is associated with the print job.
- JOB_STATUS_OFFLINE
  The printer responsible for printing the print job is offline.
- JOB_STATUS_PAPEROUT
  The printer responsible for printing the print job is out of paper.
- JOB_STATUS_PAUSED
  The print job is paused.
- JOB_STATUS_PRINTED
  The print job has been printed.
- JOB_STATUS_PRINTING
  The print job is being printed.
- JOB_STATUS_RESTART
  The print job has been restarted.
- JOB_STATUS_SPOOLING
  The print job is being spooled.
- JOB_STATUS_USER_INTERVENTION
  The printer responsible for printing the print job has suffered an error requiring the user to do something.

Priority
A value between 1 and 99 inclusive specifying the priority of the print job (99 being highest). One of the following flags can be used as well:

- DEF_PRIORITY
  Default priority.
**MIN_PRIORITY**

Minimum priority.

**MAX_PRIORITY**

Maximum priority.

**Position**

The current position of the print job in the print queue.

**TotalPages**

The total number of pages which make up the print job, if available.

**PagesPrinted**

The number of pages of the print job which have already been printed, if available.

**Submitted**

The date and time when the document was spooled into the queue. This time is in Universal Coordinated Time (UTC, a.k.a. Greenwich Mean Time, or GMT) instead of in the system's local time zone time.

### Constant Definitions

```vbnet
Const JOB_STATUS_BLOCKED_DEVQ = &H200
Const JOB_STATUS_DELETED = &H100
Const JOB_STATUS_DELETING = &H4
Const JOB_STATUS_ERROR = &H2
Const JOB_STATUS_OFFLINE = &H20
Const JOB_STATUS_PAPEROUT = &H40
Const JOB_STATUS_PAUSED = &H1
Const JOB_STATUS_PRINTED = &H80
Const JOB_STATUS_PRINTING = &H10
Const JOB_STATUS_RESTART = &H800
Const JOB_STATUS_SPOOLING = &H8
Const JOB_STATUS_USER_INTERVENTION = &H400
Const DEF_PRIORITY = 1
Const MIN_PRIORITY = 1
Const MAX_PRIORITY = 99
```

### Used By

- EnumJobs

Go back to the Structure listing.
Go back to the Reference section index.
JOB_INFO_2 Structure

Type JOB_INFO_2
   JobId As Long
   pPrinterName As String
   pMachineName As String
   pUserName As String
   pDocument As String
   pNotifyName As String
   pDatatype As String
   pPrintProcessor As String
   pParameters As String
   pDriverName As String
   pDevMode As DEVMODE
   pStatus As String
   pSecurityDescriptor As SECURITY_DESCRIPTOR
   Status As Long
   Priority As Long
   Position As Long
   StartTime As Long
   UntilTime As Long
   TotalPages As Long
   Size As Long
   Submitted As SYSTEMTIME
   Time As Long
   PagesPrinted As Long
End Type

Description & Usage

The JOB_INFO_2 structure holds information about a print job. The structure contains data which both identifies the originator of the print job and its current status in the print spooler.

Visual Basic-Specific Issues
None.

### Data Members

**JobId**
The unique numeric identifier of the print job.

**pPrinterName**
The name of the printer for which the print job is spooled.

**pMachineName**
The name of the computer which created the print job.

**pUserName**
The name of the user who owns the print job.

**pDocument**
The name of the print job (usually the document being printed).

**pNotifyName**
The name of the user who should be notified when the print job is complete or when an error occurs while printing.

**pDatatype**
The type of data used to record the print job.

**pPrintProcessor**
The name of the print processor to be used to print the print job.

**pParameters**
The parameters of the print processor.

**pDriverName**
The name of the printer driver to be used to process the print job.

**pDevMode**
Device initialization and environment data for the printer driver.

**pStatus**
The current status of the print job. If this string is empty, the print status can instead be found in Status.

**pSecurityDescriptor**
Reserved -- do not use.

**Status**
If necessary, a combination of the following flags identifying the current status of the print job:

- **JOB_STATUS_BLOCKED_DEVQ**
  The driver cannot print the print job.

- **JOB_STATUS_DELETED**
  The print job has been deleted.

- **JOB_STATUS_DELETING**
  The print job is being deleted.

- **JOB_STATUS_ERROR**
 
An error is associated with the print job.

**JOB_STATUS_OFFLINE**
The printer responsible for printing the print job is offline.

**JOB_STATUS_PAPEROUT**
The printer responsible for printing the print job is out of paper.

**JOB_STATUS_PAUSED**
The print job is paused.

**JOB_STATUS_PRINTED**
The print job has been printed.

**JOB_STATUS_PRINTING**
The print job is being printed.

**JOB_STATUS_RESTART**
The print job has been restarted.

**JOB_STATUS_SPOOLING**
The print job is being spooled.

**JOB_STATUS_USER_INTERVENTION**
The printer responsible for printing the print job has suffered an error requiring the user to do something.

**Priority**
A value between 1 and 99 inclusive specifying the priority of the print job (99 being highest). One of the following flags can be used as well:

**DEF_PRIORITY**
Default priority.

**MIN_PRIORITY**
Minimum priority.

**MAX_PRIORITY**
Maximum priority.

**Position**
The current position of the print job in the print queue.

**StartTime**
The earliest time that the print job can be printed (specified as number of minutes after midnight UTC (Universal Coordinated Time, a.k.a. Greenwich Mean Time or GMT).

**UntilTime**
The latest time that the print job can be printed (specified as number of minutes after midnight UTC (Universal Coordinated Time, a.k.a. Greenwich Mean Time or GMT).

**TotalPages**
The total number of pages which make up the print job, if available.

**Size**
The size in bytes of the print job.

**Submitted**
The date and time when the document was spooled into the queue. This time is in Universal Coordinated Time (UTC, a.k.a. Greenwich Mean Time, or GMT) instead of in the system's local time zone time.
Time
The total amount of time, in seconds, since the print job has begun printing.

PagesPrinted
The number of pages of the print job which have already been printed, if available.

Constant Definitions

Const JOB_STATUS_BLOCKED_DEVQ = &H200
Const JOB_STATUS_DELETED = &H100
Const JOB_STATUS_DELETING = &H4
Const JOB_STATUS_ERROR = &H2
Const JOB_STATUS_OFFLINE = &H20
Const JOB_STATUS_PAPEROUT = &H40
Const JOB_STATUS_PAUSED = &H1
Const JOB_STATUS_PRINTED = &H80
Const JOB_STATUS_PRINTING = &H10
Const JOB_STATUS_SPOOLING = &H8
Const JOB_STATUS_USER_INTERVENTION = &H400
Const DEF_PRIORITY = 1
Const MIN_PRIORITY = 1
Const MAX_PRIORITY = 99

Used By

EnumJobs

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
JOYCAPS Structure

Type JOYCAPS
    wMid As Integer
    wPid As Integer
    szPname As String * 32
    wXmin As Long
    wXmax As Long
    wYmin As Long
    wYmax As Long
    wZmin As Long
    wZmax As Long
    wNumButtons As Long
    wPeriodMin As Long
    wPeriodMax As Long
    wRmin As Long
    wRmax As Long
    wUmin As Long
    wUmax As Long
    wVmin As Long
    wVmax As Long
    wMaxAxes As Long
    wNumAxes As Long
    wMaxButtons As Long
    szRegKey As String * 32
    szOEMVxD As String * 240
End Type

JOYCAPS-type variables hold information about a joystick (not to be confused with the current position of the joystick). Namely, this structure holds the axes' ranges and the number of buttons the joystick has.

wMid
    The manufacturer identifier of the device.

wPid
    The product identifier of the device.

szPname
The name of the joystick's device driver.

\texttt{wXmin}

The minimum x-axis coordinate value.

\texttt{wXmax}

The maximum x-axis coordinate value.

\texttt{wYmin}

The minimum y-axis coordinate value.

\texttt{wYmax}

The maximum y-axis coordinate value.

\texttt{wZmin}

The minimum z-axis coordinate value.

\texttt{wZmax}

The maximum z-axis coordinate value.

\texttt{wNumButtons}

The number of buttons on the joystick.

\texttt{wPeriodMin}

The minimum supported polling frequency.

\texttt{wPeriodMax}

The maximum supported polling frequency.

\texttt{wRmin}

The minimum r-axis (rudder, fourth axis) coordinate value.

\texttt{wRmax}

The maximum r-axis (rudder, fourth axis) coordinate value.

\texttt{wUmin}

The minimum u-axis (fifth axis) coordinate value.

\texttt{wUmax}

The maximum u-axis (fifth axis) coordinate value.

\texttt{wVmin}

The minimum v-axis (sixth axis) coordinate value.

\texttt{wVmax}

The maximum v-axis (sixth axis) coordinate value.

\texttt{wCaps}

Zero or more flags specifying various capabilities or characteristics of the joystick:

\texttt{JOYCAPS\_HASPOV = \&H10}

Joystick provides point-of-view information.

\texttt{JOYCAPS\_HASR = \&H2}

Joystick provides rudder (fourth axis) information.

\texttt{JOYCAPS\_HASU = \&H4}

Joystick provides u-axis (fifth axis) information.

\texttt{JOYCAPS\_HASV = \&H8}

Joystick provides v-axis (sixth axis) information.

\texttt{JOYCAPS\_HASZ = \&H1}

Joystick provides z-axis information.
JOYCAPS_POV4DIR = &H20
    Joystick's point-of-view indicator supports discrete values (e.g., centered, left, right, up, down, etc.).

JOYCAPS_POVCTS = &H40
    Joystick's point-of-view indicator supports continuous degree bearings.

wMaxAxes
    The maximum number of axes supported by the joystick.

wNumAxes
    The number of axes currently used by the joystick.

wMaxButtons
    The maximum number of buttons supported by the joystick.

szRegKey
    The registry key that holds the joystick's information.

szOEMVxD
    Identifies the original equipment manufacturer (OEM) of the joystick.

Used by: joyGetDevCaps
JOYINFO Structure

Type JOYINFO
    wXpos As Long
    wYpos As Long
    wZpos As Long
    wButtons As Long
End Type

JOYINFO-type variables hold the current position of a joystick. This structure can store the positions of the x, y, and z axes, as well as the buttons pushed. Note that this structure can only receive information about buttons 1 through 4 on the joystick -- if there are more, they are ignored.

wXpos
    The current x-axis coordinate.

wYpos
    The current y-axis coordinate.

wZpos
    The current z-axis coordinate.

wButtons
    Zero or more of the following flags, specifying which buttons are being depressed:
    JOY_BUTTON1 = &H1
        Button #1 is depressed.
    JOY_BUTTON2 = &H2
        Button #2 is depressed.
    JOY_BUTTON3 = &H4
        Button #3 is depressed.
    JOY_BUTTON4 = &H8
        Button #4 is depressed.

Used by: joyGetPos
KEYBDINPUT

Type KEYBDINPUT
    wVk As Integer
    wScan As Integer
    dwFlags As Long
    time As Long
    dwExtraInfo As Long
End Type

Description & Usage

The KEYBDINPUT structure holds information about a keyboard input event. The various data members describe the exact nature of the keyboard input event. **Windows 2000**: This structure can also be used to synthesized keyboard input generated by a hardware device imitating the keyboard.

Visual Basic-Specific Issues

None.

Data Members

**wVk**

The virtual-key code of the key to simulate pressing or releasing. If `dwFlags` contains the KEYEVENTF_UNICODE tag, this must be 0.

**wScan**

If `dwFlags` contains the KEYEVENTF_UNICODE flag, this specifies the hardware scan code of the Unicode character key to simulate pressing or releasing. If that flag is not used, this must be 0.

**dwFlags**

A combination of the following flags specifying what kind of keyboard input to synthesize:

- KEYEVENTF_EXTENDEDKEY
  Prefix the scan code with a prefix byte having the value &HE0.
- KEYEVENTF_KEYUP

Prefix the scan code with a prefix byte having the value &HE0.
The key specified in bVk is being released. If this flag is not specified, the key is being pressed.

KEYEVENTF_UNICODE

Windows 2000: Use a Unicode character key generated by a non-keyboard hardware input which is imitating keyboard input.

time

The time stamp of the keyboard input event, in milliseconds. If 0, the system creates a time stamp by default.

dwExtraInfo

An additional 32-bit value associated with the keyboard event.

Constant Definitions

Const KEYEVENTF_EXTENDEDKEY = &H1
Const KEYEVENTF_KEYUP = &H2
Const KEYEVENTF_UNICODE = &H4

Used By

INPUT_TYPE

Back to the index.

---

Last Modified: August 11, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/k/keybdinput.html
LARGE_INTEGER Structure

Type LARGE_INTEGER
    LowPart As Long
    HighPart As Long
End Type

Description & Usage

The LARGE_INTEGER structure stores a 64-bit signed integer. Unsigned integers range in value from $-2^{63}$ to $2^{63} - 1$. This structure is defined primarily for languages (such as Visual Basic but including many others) which have no intrinsic support for 64-bit signed integers. The structure splits the value into two 32-bit halves: a high-order half and a low-order half. If the programming language you use does support 64-bit signed integers, you can replace the two data members of this structure with a single value (or use that data type instead of the structure altogether).

Visual Basic-Specific Issues

There is a way to use a variable of the Currency data type to represent a 64-bit integer. For more information, read the article "Faking 64-bit Integers."

Data Members

LowPart
    The 32-bit low-order half of the full 64-bit signed integer.

HighPart
    The 32-bit high-order half of the full 64-bit signed integer.

Used By

QueryPerformanceCounter, QueryPerformanceFrequency
LOGFONT Structure

Type LOGFONT

    lfHeight As Long
    lfWidth As Long
    lfEscapement As Long
    lfOrientation As Long
    lfWeight As Long
    lfItalic As Byte
    lfUnderline As Byte
    lfStrikeOut As Byte
    lfCharSet As Byte
    lfOutPrecision As Byte
    lfClipPrecision As Byte
    lfQuality As Byte
    lfPitchAndFamily As Byte
    lfFaceName As String * 32

End Type

Description & Usage

The LOGFONT structure holds information about a logical font. The various members of the structure specify properties of the logical font.

Visual Basic-Specific Issues

None.

Data Members

*lfHeight*

The height of the font's character cell, in logical units (also known as the em height). If positive, the font mapper converts this value directly into device units and matches it with the cell height of
the possible fonts. If 0, the font mapper uses a default character height. If negative, the font mapper converts the absolute value into device units and matches it with the character height of the possible fonts.

lfWidth
The average width of the font's characters. If 0, the font mapper tries to determine the best value.

lfEscapement
The angle between the font's baseline and escapement vectors, in units of \(1/10\) degrees. **Windows 95, 98:** This must be equal to lfOrientation.

lfOrientation
The angle between the font's baseline and the device's x-axis, in units of \(1/10\) degrees. **Windows 95, 98:** This must be equal to lfEscapement.

lfWeight
One of the following flags specifying the boldness (weight) of the font:

- FW_DONTCARE
  Default weight.
- FW_THIN
  Thin weight.
- FW_EXTRALIGHT
  Extra-light weight.
- FW_ULTRALIGHT
  Same as FW_EXTRALIGHT.
- FW_LIGHT
  Light weight.
- FW_NORMAL
  Normal weight.
- FW_REGULAR
  Same as FW_NORMAL.
- FW_MEDIUM
  Medium weight.
- FW_SEMIBOLD
  Semi-bold weight.
- FW_DEMIBOLD
  Same As FW_SEMIBOLD.
- FW_BOLD
  Bold weight.
- FW_EXTRABOLD
  Extra-bold weight.
- FW_ULTRABOLD
  Same as FW_EXTRABOLD.
- FW_HEAVY
  Heavy weight.
- FW_BLACK
  Same as FW_HEAVY.
**lfItalic**
A non-zero value if the font is italicized, 0 if not.

**lfUnderline**
A non-zero value if the font is underlined, 0 if not.

**lfStrikeOut**
A non-zero value if the font is striked out, 0 if not.

**lfCharSet**
Exactly one of the following flags specifying the character set of the font:
- ANSI_CHARSET
  - ANSI character set.
- ARABIC_CHARSET
  - **Windows NT, 2000**: Arabic character set.
- BALTIC_CHARSET
  - **Windows 95, 98**: Baltic character set.
- CHINESEBIG5_CHARSET
  - Chinese Big 5 character set.
- DEFAULT_CHARSET
  - Default character set.
- EASTEUROPE_CHARSET
  - **Windows 95, 98**: Eastern European character set.
- GB2312_CHARSET
  - GB2312 character set.
- GREEK_CHARSET
  - **Windows 95, 98**: Greek character set.
- HANGEUL_CHARSET
  - HANDEUL character set.
- HEBREW_CHARSET
  - **Windows NT, 2000**: Hebrew character set.
- JOHAB_CHARSET
  - **Windows 95, 98**: Johab character set.
- MAC_CHARSET
  - **Windows 95, 98**: Mac character set.
- OEM_CHARSET
  - Original equipment manufacturer (OEM) character set.
- RUSSIAN_CHARSET
  - **Windows 95, 98**: Russian character set.
- SHIFTJIS_CHARSET
  - ShiftJis character set.
- SYMBOL_CHARSET
  - Symbol character set.
- THAI_CHARSET
  - **Windows NT, 2000**: Thai character set.
- TURKISH_CHARSET
Windows 95, 98: Turkish character set.

**lfOutPrecision**

Exactly one of the following flags specifying the desired precision (closeness of the match) between the logical font ideally described by the structure and the actual logical font. This value is used by the font mapper to produce the logical font.

OUT_DEFAULT_PRECIS
- The default font mapping behavior.

OUT_DEVICE_PRECIS
- Choose a device font if there are multiple fonts in the system with the same name.

OUT_OUTLINE_PRECIS
- Windows NT, 2000: Choose a TrueType or other outline-based font.

OUT_RASTER_PRECIS
- Choose a raster font if there are multiple fonts in the system with the same name.

OUT_STRING_PRECIS
- Raster font (used for enumeration only).

OUT_STROKE_PRECIS
- Windows 95, 98: Vector font (used for enumeration only). Windows NT, 2000: TrueType, outline-based, or vector font (used for enumeration only).

OUT_TT_ONLY_PRECIS
- Choose only a TrueType font.

OUT_TT_PRECIS
- Choose a TrueType font if there are multiple fonts in the system with the same name.

**lfClipPrecision**

Exactly one of the following flags specifying the clipping precision to use when the font's characters must be clipped:

CLIP_DEFAULT_PRECIS
- The default clipping behavior.

CLIP_EMBEDDED
- This flag must be set for an embedded read-only font.

CLIP_LH_ANGLES
- The direction of any rotations is determined by the coordinate system (or else all rotations are counterclockwise).

CLIP_STROKE_PRECIS
- Raster, vector, or TrueType font (used for enumeration only).

**lfQuality**

Exactly one of the following flags specifying the output quality of the logical font as compared to the ideal font:

ANTIALIASED_QUALITY
- Windows 95, 98, NT 4.0 or later, 2000: The font is always antialiased if possible.

DEFAULT_QUALITY
- The default quality: the appearance of the font does not matter.

DRAFT_QUALITY
- The appearance of the font is less important then in PROOF_QUALITY.
**NONANTIALIASED_QUALITY**

*Windows 95, 98, NT 4.0 or later, 2000:* The font is never antialiased.

**PROOF_QUALITY**

The quality of the appearance of the font is more important than exactly matching the specified font attributes.

**lfPitchAndFamily**

A bitwise OR combination of exactly one *Pitch flag specifying the pitch of the font and exactly one *FaceFamily flag specifying the font face family of the font:

- **DEFAULT_PITCH**
  
  The default pitch.

- **FIXED_PITCH**
  
  Fixed pitch.

- **VARIABLE_PITCH**
  
  Variable pitch.

- **FF_DECORATIVE**
  
  Showy, decorative font face.

- **FF_DONTCARE**
  
  Do not care about the font face.

- **FF_MODERN**
  
  Modern font face (monospaced, sans serif font).

- **FF_ROMAN**
  
  Roman font face (proportional-width, serif font).

- **FF_SCRIPT**
  
  Script font face which imitates script handwriting.

- **FF_SWISS**
  
  Swiss font face (proportional-width, sans serif font).

**lfFaceName**

The name of the font face to use. This string must be terminated with a null character.

---

**Constant Definitions**

```vbnet
Const FW_DONTCARE = 0
Const FW_THIN = 100
Const FW_EXTRALIGHT = 200
Const FW_ULTRALIGHT = 200
Const FW_LIGHT = 300
Const FW_NORMAL = 400
Const FW_REGULAR = 400
Const FW_MEDIUM = 500
Const FW_SEMIBOLD = 600
Const FW_DEMIBOLD = 600
Const FW_BOLD = 700
```
Const FW_EXTRABOLD = 800
Const FW_ULTRABOLD = 800
Const FW_HEAVY = 900
Const FW_BLACK = 900
Const ANSI_CHARSET = 0
Const ARABIC_CHARSET = 178
Const BALTIC_CHARSET = 186
Const CHINESEBIG5_CHARSET = 136
Const DEFAULT_CHARSET = 1
Const EASTEUROPE_CHARSET = 238
Const GB2312_CHARSET = 134
Const GREEK_CHARSET = 161
Const HANGEUL_CHARSET = 129
Const HEBREW_CHARSET = 177
Const JOHAB_CHARSET = 130
Const MAC_CHARSET = 77
Const OEM_CHARSET = 255
Const RUSSIAN_CHARSET = 204
Const SHIFTJIS_CHARSET = 128
Const SYMBOL_CHARSET = 2
Const THAI_CHARSET = 222
Const TURKISH_CHARSET = 162
Const OUT_DEFAULT_PRECIS = 0
Const OUT_DEVICE_PRECIS = 5
Const OUT_OUTLINE_PRECIS = 8
Const OUT_RASTER_PRECIS = 6
Const OUT_STRING_PRECIS = 1
Const OUT_STROKE_PRECIS = 3
Const OUT_TT_ONLY_PRECIS = 7
Const OUT_TT_PRECIS = 4
Const CLIP_DEFAULT_PRECIS = 0
Const CLIP_EMBEDDED = 128
Const CLIP_LH_ANGLES = 16
Const CLIP_STROKE_PRECIS = 2
Const ANTIALIASED_QUALITY = 4
Const DEFAULT_QUALITY = 0
Const DRAFT_QUALITY = 1
Const NONANTIALIASED_QUALITY = 3
Const PROOF_QUALITY = 2
Const DEFAULT_PITCH = 0
Const FIXED_PITCH = 1
Const VARIABLE_PITCH = 2
Const FF_DECORATIVE = 80
Const FF_DONTCARE = 0
Const FF_MODERN = 48
Const FF_ROMAN = 16
Const FF_SCRIPT = 64
Const FF_SWISS = 32

Used By

CHOOSEFONT_TYPE, CreateFontIndirect, EnumFontFamiliesEx, ENUMLOGFONT,
ENUMLOGFONTEX, ICONMETRICS, NONCLIENTMETRICS, SystemParametersInfo

Go back to the Structure listing.
Go back to the Reference section index.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/logfont.html
LOGPEN Structure

Type LOGPEN
  lopnStyle As Long
  lopnWidth As POINT_TYPE
  lopnColor As Long
End Type

Description & Usage

The LOGPEN structure holds information used to identify a logical pen. The pen described by the structure is always shaped like a square.

Visual Basic-Specific Issues

None.

Data Members

lopnStyle
  One of the following flags specifying the style of the pen:
  PS_SOLID
    The pen is solid.
  PS_DASH
    The pen is dashed. The width must be less than or equal to one.
  PS_DOT
    The pen is dotted. The width must be less than or equal to one.
  PS_DASHDOT
    The pen has alternating dashes and dots. The width must be less than or equal to one.
  PS_DASHDOTDOT
    The pen has alternating dashes followed by two dots. The width must be less than or equal to one.
  PS_NULL
The pen is invisible.

**PS_INSIDEFRAME**

The pen is solid. Whenever a drawing function draws a figure inside a bounding rectangle, the dimensions of the figure are shrunk so that the entire figure, including the width of the pen, fits entirely within the bounding rectangle.

\[ lopnWidth \]

The \( x \) member of the structure identifies the width of the pen. A width of zero produces a one-pixel-wide pen no matter what. The \( y \) member of the structure is ignored.

\[ lopnColor \]

The RGB value of the color of the pen.

**Constant Definitions**

```
Const PS_SOLID = 0
Const PS_DASH = 1
Const PS_DOT = 2
Const PS_DASHDOT = 3
Const PS_DASHDOTDOT = 4
Const PS_NULL = 5
Const PS_INSIDEFRAME = 6
```

**Used By**

- CreatePenIndirect

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** October 16, 1999

This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000

Go back to the Windows API Guide home page.

E-mail: [vbapi@vbapi.com](mailto:vbapi@vbapi.com) [Send Encrypted E-Mail](#)

This page is at [http://www.vbapi.com/ref/l/logpen.html](http://www.vbapi.com/ref/l/logpen.html)
MEMORYSTATUS Structure

Type MEMORYSTATUS
  dwLength As Long
  dwMemoryLoad As Long
  dwTotalPhys As Long
  dwAvailPhys As Long
  dwTotalPageFile As Long
  dwAvailPageFile As Long
  dwTotalVirtual As Long
  dwAvailVirtual As Long
End Type

Description & Usage

The MEMORYSTATUS structure holds information about the computer's memory. The structure stores the amounts of total and available physical memory and virtual memory.

Visual Basic-Specific Issues

None.

Data Members

\textit{dwLength}

The size in bytes of the structure.

\textit{dwMemoryLoad}

\textbf{Windows 95, 98, NT}: The percentage of approximately the last 1000 pages of physical memory that is in use. \textbf{Windows 2000}: The percentage of total physical memory in use.

\textit{dwTotalPhys}

The number of bytes of total physical memory.

\textit{dwAvailPhys}

The number of bytes of available physical memory.
**dwTotalPageFile**

The largest possible size, in bytes, of the paging file. This is not necessarily the current size of the paging file.

**dwAvailPageFile**

The amount of space, in bytes, currently available in the paging file.

**dwTotalVirtual**

The number of bytes of total virtual memory.

**dwAvailVirtual**

The number of bytes of available virtual memory.

---

**Used By**

GlobalMemoryStatus

[Back to the Structure list.](#)

[Back to the Reference section.](#)
MEMORYSTATUSEX Structure

Type MEMORYSTATUSEX
    dwLength As Long
    dwMemoryLoad As Long
    ullTotalPhys As ULARGE_INTEGER
    ullAvailPhys As ULARGE_INTEGER
    ullTotalPageFile As ULARGE_INTEGER
    ullAvailPageFile As ULARGE_INTEGER
    ullTotalVirtual As ULARGE_INTEGER
    ullAvailVirtual As ULARGE_INTEGER
    ullAvailExtendedVirtual As ULARGE_INTEGER
End Type

Description & Usage

The MEMORYSTATUSEX structure holds information about the computer's memory. The structure stores the amounts of total and available physical memory and virtual memory. All values for amounts of memory are stored as 64-bit integers.

Visual Basic-Specific Issues

None.

Data Members

\textit{dwLength} \\
The size in bytes of the structure.

\textit{dwMemoryLoad} \\
The percentage of total physical memory in use.

\textit{ullTotalPhys} \\
The number of bytes of total physical memory.
**ullAvailPhys**

The number of bytes of available physical memory.

**ullTotalPageFile**

The largest possible size, in bytes, of the paging file. This is not necessarily the current size of the paging file.

**ullAvailPageFile**

The amount of space, in bytes, currently available in the paging file.

**ullTotalVirtual**

The number of bytes of total virtual memory.

**ullAvailVirtual**

The number of bytes of available virtual memory.

**ullAvailExtendedVirtual**

The number of bytes of unreserved and uncommitted memory in the extended portion of virtual memory.

### Used By

- [GlobalMemoryStatusEx](http://www.vbapi.com/ref/m/memorystatus.html)

---

**Last Modified:** April 16, 2000

This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/m/memorystatusex.html) Revised October 29, 2000

[Go back to the Windows API Guide home page.](http://www.vbapi.com/ref/m/memorystatusex.html)

E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at [http://www.vbapi.com/ref/m/memorystatusex.html](http://www.vbapi.com/ref/m/memorystatusex.html)
MENUITEMINFO Structure

Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type

Description & Usage

The MENUITEMINFO structure holds information that describes a menu item. This description includes the text of the item, its state (enabled, gray, etc.), and other things relating to its appearance.

Visual Basic-Specific Issues

If you need to set dwTypeData to a handle to a bitmap, a slight modification to the above structure definition must be made. Change the type of the dwTypeData data member from "As String" to "As Long". With that change made, it can now be set to a bitmap handle.

Data Members

cbSize
    The size, in bytes, of the structure.

fMask
    A combination of the following flags specifying which parts of the structure to use. Any other
data members not specified by a flag will be ignored.

**MIIM_STATE**
Use the *fState* data member.

**MIIM_ID**
Use the *wID* data member.

**MIIM_SUBMENU**
Use the *hSubMenu* data member.

**MIIM_CHECKMARKS**
Use the *hbmpChecked* and *hbmpUnchecked* data members.

**MIIM_DATA**
Use the *dwItemData* data member.

**MIIM_TYPE**
Use the *dwTypeData* and *cch* data members.

*fType*
A combination of the following flags specifying the type of menu item the structure describes:

**MFT_BITMAP**
The menu item is displayed using a bitmap. *dwTypeData* is a handle to the bitmap to use, and *cch* is ignored. This flag cannot be combined with MFT_SEPARATOR or MFT_STRING.

**MFT_MENUBARBREAK**
Place this item on a new row (for a menu bar) or column (for a drop-down menu, submenu, or context menu). If starting a new column, separate the two columns by a vertical line.

**MFT_MENUBREAK**
Same as MFT_MENUBARBREAK, but do not draw a line separating the two columns.

**MFT_OWNERDRAW**
Make the menu item owner-drawn. This makes the owning window completely responsible for drawing the menu item whenever necessary.

**MFT_RADIOCHECK**
When checked, place a radio-button mark instead of a check mark next to the menu item. This signifies a radio-button-type menu item.

**MFT_RIGHTJUSTIFY**
For menu bars only, right-justify this item and all other items after it.

**MFT_RIGHTORDER**
**Windows 95, 98, 2000:** The menus cascade from right to left, supporting languages that are read from right to left.

**MFT_SEPARATOR**
The menu item is a separator bar. *dwTypeData* and *cch* are ignored. This flag cannot be combined with MFT_BITMAP or MFT_STRING.

**MFT_STRING**
The menu item is represented by a text string (which may or may not be checked or have a submenu attached). *dwTypeData* is the string to display for the menu item, and *cch* is the string's length. This flag cannot be combined with MFT_BITMAP or MFT_SEPARATOR.
fState
A combination of the following flags specifying the menu item's current state:

MFS_CHECKED
A check mark (or radio-button mark) appears next to the menu item.

MFS_DEFAULT
This item is the default selection for the menu.

MFSDISABLED
The menu item is disabled. Although it is displayed normally, the user cannot select it.

MFS_ENABLED
The menu item is enabled.

MFS_GRAYED
The menu item is grayed out. The user cannot select it.

MFS_HILITE
The menu item currently has the selection highlight.

MFS_UNCHECKED
No check mark (or radio-button mark) appears next to the menu item.

MFS_UNHILITE
The menu item does not currently have the selection highlight.

wID
The unique 16-bit identifier of the menu item.

hSubMenu
A handle to the submenu that opens when the menu item is selected.

hbmpChecked
A handle to the bitmap to display next to the menu item when the item is checked. The bitmap's colors will be inverted when the selection highlight moves over the item.

hbmpUnchecked
A handle to the bitmap to display next to the menu item when the item is not checked. The bitmap's colors will be inverted when the selection highlight moves over the item.

dwItemData
Program-defined information to associate with the menu item. This information is not used by Windows.

dwTypeData
If fType contains MFT_STRING, this is the string to display for the menu item. If fType contains MFT_BITMAP, this is a handle to the bitmap used to represent the menu item.

cch
If dwTypeData is a string, this is the number of characters in that string.

Constant Definitions

Const MIIM_STATE = &H1
Const MIIM_ID = &H2
Const MIIM_SUBMENU = &H4
Const MIIM_CHECKMARKS = &H8
Const MIIM_DATA = &H20
Const MIIM_TYPE = &H10
Const MFT_BITMAP = &H4
Const MFT_MENUBARBREAK = &H20
Const MFT_MENUBREAK = &H40
Const MFT_OWNERDRAW = &H100
Const MFT_RADIOCHECK = &H200
Const MFT_RIGHTJUSTIFY = &H4000
Const MFT_RIGHTORDER = &H2000
Const MFT_SEPARATOR = &H800
Const MFT_STRING = &H0
Const MFS_CHECKED = &H8
Const MFS_DEFAULT = &H1000
Const MFS_DISABLED = &H2
Const MFS_ENABLED = &H0
Const MFS_GRAYED = &H1
Const MFS_HILITE = &H80
Const MFS_UNCHECKED = &H0
Const MFS_UNHILITE = &H0

Used By

GetMenuItemInfo, InsertMenuItem, SetMenuItemInfo

Back to the Structure list.
Back to the Reference section.

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/menuteminfo.html
MINIMIZEDMETRICSTYPE

cbSize As Long
iWidth As Long
iHorzGap As Long
iVertGap As Long
iArrange As Long
End Type

MINIMIZEDMETRICSTYPE-type variable hold information about the metrics of minimized windows. The metrics of minimized windows specify various properties that Windows uses to work with all minimized windows. The structure stores these metrics.

\textit{cbSize} \\
\textit{iWidth} \\
\textit{iHorzGap} \\
\textit{iVertGap} \\
\textit{iArrange}

\textit{cbSize} \\
The size in bytes of the structure.

\textit{iWidth} \\
The width in pixels of minimized windows.

\textit{iHorzGap} \\
The horizontal space in pixels between arranged minimized windows.

\textit{iVertGap} \\
The vertical space in pixels between arranged minimized windows.

\textit{iArrange} \\
Exactly two of the following flags specifying the method used to display minimized windows. One flag specifies a starting position for the minimized icons and the other specifies the direction in which new ones are added:

ARW_BOTTOMLEFT = 0
Start placing the icons in the bottom-left corner of the screen.

ARW_BOTTOMRIGHT = 1
Start placing the icons in the bottom-right corner of the screen.

ARW_DOWN = 4
Add new icons below existing ones.

ARW_HIDE = 8
Do not place the icons anywhere on the screen (i.e., hide them).

ARW_LEFT = 0
Add new icons to the left of existing ones.
ARW_RIGHT = 4
  Add new icons to the right of existing ones.
ARW_STARTRIGHT = 1
  Same as ARW_BOTTOMRIGHT.
ARW_STARTTOP = 2
  Same as ARW_TOPLEFT.
ARW_TOPLEFT = 2
  Start placing the icons in the top-left corner of the screen.
ARW_TOPRIGHT = 3
  Start placing the icons in the top-right corner of the screen.
ARW_UP = 0
  Add new icons above existing ones.

**Used by:** SystemParametersInfo
MOUSEINPUT Structure

Type MOUSEINPUT
    dx As Long
    dy As Long
    mouseData As Long
    dwFlags As Long
    time As Long
    dwExtraInfo As Long
End Type

Description & Usage

The MOUSEINPUT structure holds information about a mouse input event. The various data members identify the nature of the input.

Visual Basic-Specific Issues

None.

Data Members

dx

Specifies either the x-coordinate of absolute mouse movement or the amount of relative movement along the x-axis. For relative motion, positive values move right and negative values move left.

dy

Specifies either the y-coordinate of absolute mouse movement or the amount of relative movement along the y-axis. For relative motion, positive values move down and negative values move up.

mouseData

Windows NT, 2000: If dwFlags contains MOUSEEVENTF_WHEEL, this specifies the amount of wheel movement, in integer multiples of WHEEL_DATA. Positive values mean forward
(away) rotation, and negative values mean backwards (toward) rotation. **Windows 2000:** If _dwFlags_ contains either MOUSEEVENTF_XDOWN or MOUSEEVENTF_XUP, this is a combination of the following flags specifying which X buttons have been pressed or released:

- **XBUTTON1**
  - The first X button was pressed or released.
- **XBUTTON2**
  - The second X button was pressed or released.

**dwFlags**

A combination of the following flags specifying which mouse input information the event describes. Only specify button status information for those which have changed. Note that scroll wheel movement and X button status cannot be simultaneously specified because they both use the _mouseData_ data member.

- **MOUSEEVENTF_ABSOLUTE**
  - The _dx_ and _dy_ data members contain absolute mouse coordinates. In the coordinate system used by the function, the screen's upper-left corner has coordinates (0,0) and the lower-right corner has coordinates (65535,65535), regardless of the actual screen size. If this flag is not set, _dx_ and _dy_ contain relative coordinates, whose actual amount of movement depends on the current mouse speed and acceleration settings.
- **MOUSEEVENTF_LEFTDOWN**
  - The left button was pressed.
- **MOUSEEVENTF_LEFTUP**
  - The left button was released.
- **MOUSEEVENTF_MIDDLEDOWN**
  - The middle button was pressed.
- **MOUSEEVENTF_MIDDLEUP**
  - The middle button was released.
- **MOUSEEVENTF_MOVE**
  - The mouse moved. The _dx_ and _dy_ data members specify the amount or location of the movement.
- **MOUSEEVENTF_RIGHTDOWN**
  - The right button was pressed.
- **MOUSEEVENTF_RIGHTUP**
  - The right button was released.
- **MOUSEEVENTF_WHEEL**
  - **Windows NT, 2000:** The scroll wheel has moved. The _dwData_ data member specifies the amount of movement.
- **MOUSEEVENTF_XDOWN**
  - **Windows 2000:** An X button was pressed. The _dwData_ parameter identifies which X buttons.
- **MOUSEEVENTF_XUP**
  - **Windows 2000:** An X button was released. The _dwData_ parameter identifies which X buttons.
The time stamp of the mouse input event, in milliseconds. If this is 0, the system provides a time stamp.

*dwExtraInfo*

An additional 32-bit value associated with the mouse event.

## Constant Definitions

```plaintext
Const MOUSEEVENTF_ABSOLUTE  = &H8000
Const MOUSEEVENTF_LEFTDOWN  = &H2
Const MOUSEEVENTF_LEFTUP    = &H4
Const MOUSEEVENTF_MIDDLEDOWN= &H20
Const MOUSEEVENTF_MIDDLEUP  = &H40
Const MOUSEEVENTF_MOVE      = &H1
Const MOUSEEVENTF_RIGHTDOWN = &H8
Const MOUSEEVENTF_RIGHTUP   = &H10
Const MOUSEEVENTF_WHEEL     = &H80
Const MOUSEEVENTF_XDOWN     = &H100
Const MOUSEEVENTF_XUP       = &H200
Const WHEEL_DELTA           = 120
Const XBUTTON1             = &H1
Const XBUTTON2             = &H2
```

## Used By

**INPUT_TYPE**

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** August 11, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/m/mouseinput.html](http://www.vbapi.com/ref/m/mouseinput.html)
MOUSEKEYS Structure

Type MOUSEKEYS
    cbSize As Long
    dwFlags As Long
    iMaxSpeed As Long
    iT imeToMaxSpeed As Long
    iCtrlSpeed As Long
    dwReserved1 As Long
    dwReserved2 As Long
End Type

MOUSEKEYS-type variables store information about the MouseKeys accessibility feature. MouseKeys allows the mouse cursor to be moved by the numeric keypad in addition to an actual mouse. The structure stores the settings about the MouseKeys accessibility feature.

cbSize
The size in bytes of the structure.
dwFlags
Zero or more of the following flags specifying various settings and properties of MouseKeys:
MKF_AVAILABLE = &H2
    The MouseKeys feature is available.
MKF_CONFIRMHOTKEY = &H8
    Win 95/98 only: Open a confirmation dialog box when the user activates MouseKeys via the hot key.
MKF_HOTKEYACTIVE = &H4
    Allow the user to toggle MouseKeys via the hot key: press Left Alt, Left Shift, and Num Lock simultaneously.
MKF_HOTKEYSOUND = &H10
    Play a sound when the user toggles MouseKeys via the hot key.
MKF_INDICATOR = &H20
    Win 95/98 only: Display an icon in the system tray while MouseKeys is on.
MKF_MODIFIERS = &H40
    Win 95/98 only: Multiply the mouse speed by iCtrlSpeed while the Ctrl key is held.
MKF_MOUSEKEYSON = &H1
    The MouseKeys feature is currently on.
MKF_REPLACENUMBERS = &H80

*Win 95/98 only:* Only let the numeric keypad control the mouse cursor while Num Lock is on. If this flag is not set, only let the numeric keypad control the mouse cursor while Num Lock is off.

**iMaxSpeed**
The maximum speed of the mouse cursor. In Win NT, this must be between 10 and 360 inclusive. In Win 95/98, there is no limiting range.

**iTimeToMaxSpeed**
The length of time, in milliseconds, before the mouse cursor begins moving at the maximum speed. This must be between 1000 and 5000 inclusive.

**iCtrlSpeed**
*Win 95/98:* The multiplier to apply to the mouse speed when the Ctrl key is held. *Win NT:* Reserved -- set to 0.

**dwReserved1**
Reserved -- set to 0.

**dwReserved2**
Reserved -- set to 0.

**Used by:** [SystemParametersInfo](http://www.vbapi.com/ref/m/mousekeys.html)

[Back to the index](http://www.vbapi.com/ref/m/mousekeys.html)
MSGBOXPARAMS Structure

Type MSGBOXPARAMS
    cbSize As Long
    hwndOwner As Long
    hInstance As Long
    lpszText As String
    lpszCaption As String
    dwStyle As Long
    lpszIcon As Long
    dwContextHelpId As Long
    lpfnMsgBoxCallback As Long
    dwLanguageId As Long
End Type

Description & Usage

The MSGBOXPARAMS structure stores the options used to create a message box. The information held by the structure specifies numerous settings used to determine the presentation and functionality of the message box.

Visual Basic-Specific Issues

To specify a pointer to the MsgBoxCallback callback function for the lpfnMsgBoxCallback data member, a dummy function must be used. Because the AddressOf operator is only valid inside a function call's argument list, a program-defined dummy function that simply returns the value passed to it is necessary to set the data member. For an example of such a dummy function, see the example for the MessageBoxIndirect function.

Data Members

*cbSize*

The size in bytes of the structure.
**hwndOwner**

A handle to the window which owns the message box.

**hInstance**

A handle to the instance of the application calling the function.

**lpszText**

The text to display in the body of the message box.

**lpszCaption**

The text to display in the title bar of the message box.

**dwStyle**

A combination of various flags specifying the behavior and appearance of the message box. The available flags are grouped according to function. If no flags in a certain group are specified, the default is used.

Use one of the following flags to specify which buttons to display in the message box. (Note that MB_HELP can be combined with any of the other flags.)

- **MB_ABORTRETRYIGNORE**
  The message box contains the Abort, Retry, and Ignore buttons.

- **MBCancelarTRYCONTINUE**
  **Windows 2000:** The message box contains the Cancel, Try Again, and Continue buttons. This is meant to replace the MB_ABORTRETRYIGNORE flag.

- **MB_HELP**
  **Windows 95, 98, NT 4.0 or later, 2000:** Add the Help button to the message box. When the user clicks the Help button, the `WM_HELP` message is sent to the owner of the message box (specified by the `hWnd` parameter). This flag can only be used by combining it with another button flag (i.e., the Help button cannot appear alone).

- **MB_OK**
  The message box contains the OK button. This is the default.

- **MB_OKCANCEL**
  The message box contains the OK and Cancel buttons.

- **MB_RETRycANCEL**
  The message box contains the Retry and Cancel buttons.

- **MB_YESNO**
  The message box contains the Yes and No buttons.

- **MB_YESNOCANCEL**
  The message box contains the Yes, No, and Cancel buttons.

Use one of the following flags to specify which icon to display in the message box:

- **MB_ICONASTERISK, MB_ICONINFORMATION**
  Display the information icon: a lowercase letter "i" inside a blue circle.

- **MB_ICONERROR, MB_ICONHAND, MB_ICONSTOP**
  Display the stop-sign icon in the message box.

- **MB_ICONEXCLAMATION, MB_ICONWARNING**
  Display the exclamation-point icon in the message box.
MB_ICONQUESTION
Display the question-mark icon in the message box.

MB_USERICON
Use the icon specified by the lpszIcon data member instead of a predefined system icon.

Use one of the following flags to specify which button is selected by default:

MB_DEFAULTBUTTON1
The first button is the default. This is the default.

MB_DEFAULTBUTTON2
The second button is the default.

MB_DEFAULTBUTTON3
The third button is the default.

MB_DEFAULTBUTTON4
The fourth button is the default.

Use one of the following flags to specify the modality of the message box:

MB_APPLMODAL
The message box is application-modal. The user cannot switch to any other windows owned by the application until he or she first closes the message box. This is the default.

MB_SYSTEMMODAL
The message box is system-modal. The user cannot switch to any other windows until he or she first closes the message box.

MB_TASKMODAL
The message box is thread-modal. The user cannot switch to any other windows owned by the calling thread until he or she first closes the message box.

Use zero or more of the following flags to specify other options for the message box:

MB_DEFAULT_DESKTOP_ONLY
Windows NT, 2000: Same as MB_SERVICE_NOTIFICATION, except that the system will display the message box only on the interactive window station's default desktop.

MB_RIGHT
The text in the message box is right-justified.

MB_RTLREADING
Display the message text and caption using right-to-left reading order if desired by the system language.

MB_SETFOREGROUND
Make the message box the foreground window.

MB_TOPMOST
Make the message box a topmost window.

MB_SERVICE_NOTIFICATION
Windows NT 4.0 or later, 2000: The calling thread is a service notifying the user of an event. The hWnd parameter must be 0.

MB_SERVICE_NOTIFICATION_NT3X
Windows NT 3.1 through 3.51: Same as MB_SERVICE_NOTIFICATION. The value of
this flag changed with the release of NT 4.0.

`lpszIcon`

The icon resource to use for the message box's icon. This can be specified either by an integer resource identifier or by a pointer to a string equal to the resource's name. Either way, the resource must be owned by the instance specified by `hInstance`.

`dwContextHelpId`

The help context ID which identifies the appropriate help topic to display if the user clicks the Help button. This information is passed to the callback function specified by `lpfnMsgBoxCallback`.

`lpfnMsgBoxCallback`

A pointer to the `MsgBoxCallback` callback function to process help events for the message box. This function is invoked whenever the user clicks the Help button. If this is 0 and the user clicks the Help button, a `WM_HELP` message is sent to the message box's owner.

`dwLanguageId`

The language ID, retrieved by the `MAKELANGID` macro, which identifies the language to use to display the message box's buttons. This language must already be installed on the computer.

**Constant Definitions**

```
Const MB_ABORTRETRYIGNORE = &H2
Const MB_CANCELTRYCONTINUE = &H2
Const MB_HELP = &H4000
Const MB_OK = &H0
Const MB_OKCANCEL = &H1
Const MB_RETRYCANCEL = &H5
Const MB_YESNO = &H4
Const MB_YESNOCANCEL = &H3
Const MB_ICONASTERISK = &H40
Const MB_ICONERROR = &H10
Const MB_ICONEXCLAMATION = &H30
Const MB_ICONHAND = &H10
Const MB_ICONINFORMATION = &H40
Const MB_ICONQUESTION = &H20
Const MB_ICONSTOP = &H10
Const MB_ICONWARNING = &H30
Const MB_USERICON = &H80
Const MB_DEFBUTTON1 = &H0
Const MB_DEFBUTTON2 = &H100
Const MB_DEFBUTTON3 = &H200
Const MB_DEFBUTTON4 = &H300
Const MB_APPLMODAL = &H0
Const MB_SYSTEMMODAL = &H1000
```
Const MB_TASKMODAL = &H2000
Const MB_DEFAULT_DESKTOP_ONLY = &H20000
Const MB_RIGHT = &H80000
Const MBRTLREADING = &H100000
Const MBSETFOREGROUND = &H10000
'Const MB_TOPMOST = ???
'Const MB_SERVICE_NOTIFICATION = ???
'Const MB_SERVICE_NOTIFICATION_NT3X = ???

Used By

MessageBoxIndirect

Back to the Structure list.
Back to the Reference section.

Last Modified: February 11, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/msgboxparams.html
MULTIKEYHELP Structure

Type MULTIKEYHELP
   mkSize As Long
   mkKeylist As Byte
   szKeyphrase As String * 255  ' can actually be any length
End Type

Description & Usage

The MULTIKEYHELP structure stores one or more keywords to search for in a Windows Help file as well as the identifier of the alternate keyword table to search. Note that *szKeyphrase* can be of any length needed -- 255 bytes is just one possible value.

Visual Basic-Specific Issues

None.

Data Members

*mkSize*  
The size in bytes of the structure.

*mkKeylist*  
The identifier of the alternate keyword table to search.

*szKeyphrase*  
A null-terminated string holding one or more keywords to search for. If using more than one keyword, separate each keyword with a semicolon.

Used By

WinHelp
NEWTEXTMETRIC Structure

```vbnet
Type NEWTEXTMETRIC
    tmHeight As Long
    tmAscent As Long
    tmDescent As Long
    tmInternalLeading As Long
    tmExternalLeading As Long
    tmAveCharWidth As Long
    tmMaxCharWidth As Long
    tmWeight As Long
    tmOverhang As Long
    tmDigitizedAspectX As Long
    tmDigitizedAspectY As Long
    tmFirstChar As Byte
    tmLastChar As Byte
    tmDefaultChar As Byte
    tmBreakChar As Byte
    tmItalic As Byte
    tmUnderlined As Byte
    tmStruckOut As Byte
    tmPitchAndFamily As Byte
    tmCharSet As Byte
    ntmFlags As Long
    ntmSizeEM As Long
    ntmCellHeight As Long
    ntmAveWidth As Long
End Type
```

Description & Usage

The NEWTEXTMETRIC structure holds the text metrics of a font. The structure describes many physical attributes of the font. Unless otherwise specified, all size measurements of the font stored in the structure are in the logical units of the device the font is being used on, so the same font may have a different "scale" if used on a different device.
Visual Basic-Specific Issues

None.

Data Members

tmHeight
The height of the font's characters (equal to the sum of tmAscent and tmDescent).

tmAscent
The ascent (units above the base line) of the font's characters.

tmDescent
The descent (units below the base line) of the font's characters.

tmInternalLeading
The amount of leading space inside the bounds of tmHeight where accent marks or other diacritical marks may appear.

tmExternalLeading
The amount of leading space placed between rows of text in which the font does not draw.

tmAveCharWidth
The average width of the font's characters, usually identified as the width of the "x" character. This value does not include extra space required for bold or italic characters.

tmMaxCharWidth
The width of the widest character of the font. This value does not include extra space required for bold or Italics.

tmWeight
One of the following flags specifying the boldness (weight) of the font:

FW_DONTCARE
Default weight.

FW_THIN
Thin weight.

FW_EXTRALIGHT
Extra-light weight.

FW_ULTRALIGHT
Same as FW_EXTRALIGHT.

FW_LIGHT
Light weight.

FW_NORMAL
Normal weight.

FW_REGULAR
Same as FW_NORMAL.

FW_MEDIUM
Medium weight.
FW_SEMIBOLD
   Semi-bold weight.
FW_DEMIBOLD
   Same As FW_SEMIBOLD.
FW_BOLD
   Bold weight.
FW_EXTRABOLD
   Extra-bold weight.
FW_ULTRABOLD
   Same as FW_EXTRABOLD.
FW_HEAVY
   Heavy weight.
FW_BLACK
   Same as FW_HEAVY.

tmOverhang
   The extra width per string added to the font when synthesizing attributes such as boldface and italics. For a boldface effect, this is the distance by which the overstrike is offset. For an italics effect, this is the amount that the top of the character is sheared past the bottom of the character.

tmDigitizedAspectX
   The horizontal aspect of the device for which the font was designed.

tmDigitizedAspectY
   The vertical aspect of the device for which the font was designed.

tmFirstChar
   The value of the first character defined in the font.

tmLastChar
   The value of the last character defined in the font.

tmDefaultChar
   The value of the character to substitute for characters not present in the font.

tmBreakChar
   The value of the character to be used for work breaks for text justification.

tmItalic
   If zero, the font is not an italic font. If a non-zero value, the font is an italic font.

tmUnderlined
   If zero, the font is not an underlined font. If a non-zero value, the font is an underlined font.

tmStruckOut
   If zero, the font is not a strikeout font. If a non-zero value, the font is a strikeout font.

tmPitchAndFamily
   A bitwise OR combination of exactly one *_PITCH flag specifying the pitch of the font and exactly one FF_* flag specifying the font face family of the font:
   DEFAULT_PITCH
      The default pitch.
   FIXED_PITCH
      Fixed pitch.
VARIABLE_PITCH
  Variable pitch.
FF_DECORATIVE
  Showy, decorative font face.
FF_DONTCARE
  Do not care about the font face.
FF_MODERN
  Modern font face (monospaced, sans serif font).
FF_ROMAN
  Roman font face (proportional-width, serif font).
FF_SCRIPT
  Script font face which imitates script handwriting.
FF_SWISS
  Swiss font face (proportional-width, sans serif font).

tmCharSet
  One of the following flags identifying the character set of the font:
ANSI_CHARSET
  ANSI character set.
ARABIC_CHARSET
BALTIC_CHARSET
  Windows 95, 98: Baltic character set.
CHINESEBIG5_CHARSET
  Chinese Big 5 character set.
DEFAULT_CHARSET
  Default character set.
EASTEUROPE_CHARSET
  Windows 95, 98: Eastern European character set.
GB2312_CHARSET
  GB2312 character set.
GREEK_CHARSET
  Windows 95, 98: Greek character set.
HANGEUL_CHARSET
  HANDEUL character set.
HEBREW_CHARSET
JOHAB_CHARSET
  Windows 95, 98: Johab character set.
MAC_CHARSET
  Windows 95, 98: Mac character set.
OEM_CHARSET
  Original equipment manufacturer (OEM) character set.
RUSSIAN_CHARSET
**Windows 95, 98**: Russian character set.

SHIFTJIS_CHARSET
ShiftJis character set.

SYMBOL_CHARSET
Symbol character set.

THAI_CHARSET
**Windows NT, 2000**: Thai character set.

TURKISH_CHARSET
**Windows 95, 98**: Turkish character set.

**ntmFlags**
Zero or more of the following flags identifying various properties or attributes of the font:

- NTM_BOLD
  The font is a bold font.

- NTM_DSIG
  **Windows 2000**: The font has a digital signature, which allows the font to be traced and ensures that the font has not been corrupted.

- NTM_ITALIC
  The font is an italic font.

- NTM_MULTIPLEMASTER
  **Windows 2000**: The font is a Multiple Master font.

- NTM_NONNEGATIVE_AC
  **Windows 2000**: No glyph in the font at any size as a negative A or C space.

- NTM_PS_OPENTYPE
  **Windows 2000**: The font is a PostScript OpenType font.

- NTM_REGULAR
  The font is a regular font.

- NTM TT_OPENTYPE
  **Windows 2000**: The font is a TrueType OpenType font.

- NTM_TYPE1
  **Windows 2000**: The font is a Type 1 font.

**ntmSizeEM**
The size of the em square for the font, measured in notional units.

**ntmCellHeight**
The height of the font, measured in notional units.

**ntmAvgWidth**
The average width of the characters in the font, measured in notional units.

## Constant Definitions

Const FW_DONTCARE = 0
Const FW_THIN = 100
Const FW_EXTRALIGHT = 200
Const FW_ULTRALIGHT = 200
Const FW_LIGHT = 300
Const FW_NORMAL = 400
Const FW_REGULAR = 400
Const FW_MEDIUM = 500
Const FW_SEMIBOLD = 600
Const FW_DEMIBOLD = 600
Const FW_BOLD = 700
Const FW_EXTRABOLD = 800
Const FW_ULTRABOLD = 800
Const FW_HEavy = 900
Const FW_BLACK = 900
Const DEFAULT_PITCH = 0
Const FIXED_PITCH = 1
Const VARIABLE_PITCH = 2
Const FF_DECORATIVE = 80
Const FF_DONTCARE = 0
Const FF_ROMAN = 16
Const FF_SCRIPT = 64
Const FF_SWISS = 32
Const ANSI_CHARSET = 0
Const ARABIC_CHARSET = 178
Const BALTIC_CHARSET = 186
Const CHINESEBIG5_CHARSET = 136
Const DEFAULT_CHARSET = 1
Const EASTEUROPE_CHARSET = 238
Const GB2312_CHARSET = 134
Const GREEK_CHARSET = 161
Const HANGEUL_CHARSET = 129
Const HEBREW_CHARSET = 177
Const JOHAB_CHARSET = 130
Const MAC_CHARSET = 77
Const OEM_CHARSET = 255
Const RUSSIAN_CHARSET = 204
Const SHIFTJIS_CHARSET = 128
Const SYMBOL_CHARSET = 2
Const THAI_CHARSET = 222
Const TURKISH_CHARSET = 162
Const NTM_BOLD = &H20
Const NTM_DSIG = &H100000
Const NTM_ITALIC = &H1
Const NTM_MULTIPLEMASTER = &H40000
Const NTM_NONNEGATIVE_AC = &H10000
Windows API Guide: NEWTEXTMETRIC Structure

Const NTM_PS_OPENTYPE = &H20000
Const NTM_REGULAR = &H100
Const NTM_TYPE1 = &H80000

Used By

EnumFontFamProc, NEWTEXTMETRICEX

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: October 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/n/newtextmetric.html
NEWTEXTMETRICEX Structure

Type NEWTEXTMETRICEX
  ntmTm As NEWTEXTMETRIC
  ntmFontSig As FONTSIGNATURE
End Type

Description & Usage

The NEWTEXTMETRICEX structure holds information describing the physical attributes of a font. In addition to storing the metrics of the font, the structure also identifies the font signature.

Visual Basic-Specific Issues

None.

Data Members

ntmTm
  The text metrics of the font.
ntmFontSig
  The font signature of the font.

Used By

EnumFontFamExProc

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
NONCLIENTMETRICS Structure

Type NONCLIENTMETRICS
    cbSize As Long
    iBorderWidth As Long
    iScrollWidth As Long
    iScrollHeight As Long
    iCaptionWidth As Long
    iCaptionHeight As Long
    lfCaptionFont As LOGFONT
    iSMCaptionWidth As Long
    iSMCaptionHeight As Long
    lfSMCaptionFont As LOGFONT
    iMenuWidth As Long
    iMenuHeight As Long
    lfMenuFont As LOGFONT
    lfStatusFont As LOGFONT
    lfMessageFont As LOGFONT
End Type

NONCLIENTMETRICS-type variables store information about the metrics associated with the non-client areas of windows. A window's non-client area includes its title bar, menu bar, and border -- i.e., the things around its area. The metrics specify various properies shared by all non-client areas. Note that all widths and heights referred to in the structure are measured in pixels.

\[ cbSize \]
The size in bytes of the structure.

\[ iBorderWidth \]
The thickness of a window's sizing border.

\[ iScrollWidth \]
The width of a standard vertical scroll bar.

\[ iScrollHeight \]
The height of a standard horizontal scroll bar.

\[ iCaptionWidth \]
The width of a caption button.
iCaptionHeight
  The height of a caption button.

lfCaptionFont
  Information about the logical font used to display text in a window's caption (title bar).

iSMCaptionWidth
  The width of a small caption button.

iSMCaptionHeight
  The height of a small caption button.

iSMCaptionFont
  Information about the logical font used to display text in a small caption.

iMenuWidth
  The width of a menu bar button.

iMenuHeight
  The height of a menu bar button.

lfMenuFont
  Information about the logical font used to display text in a menu bar.

lfStatusFont
  Information about the logical font used to display text in a status bar.

lfMessageFont
  Information about the logical font used to display text in message boxes.

**Used by:** SystemParametersInfo
NOTIFYICONDATA Structure

Type NOTIFYICONDATA
    cbSize As Long
    hWnd As Long
    uID As Long
    uFlags As Long
    uCallbackMessage As Long
    hIcon As Long
    szTip As String * 64 ' Windows 2000: make this String * 128
' The following data members are only valid in Windows 2000!
' (uncomment the following lines to use them)
' dwState As Long
' dwStateMask As Long
' szInfo As String * 256
' uTimeoutOrVersion As Long
' szInfoTitle As String * 64
' dwInfoFlags As Long
End Type

Description & Usage

The NOTIFYICONDATA structure stores information used to communicate with an icon in the system tray. The structure holds data that both identifies and describes settings for the icon in question. Windows 2000 expands significantly on this structure, adding multiple data members not available in previous versions of Windows.

Visual Basic-Specific Issues

Windows 2000: Officially, the uTimeoutOrVersion data member is actually two separate data members, uTimeout and uVersion, which occupy the same space within the structure. Because Visual Basic does not support unions like C++ does, it is necessary to assign a single name to this data member. Nevertheless, using this pseudonym involves no loss of functionality.
Data Members

\textit{cbSize}

The size in bytes of the structure.

\textit{hWnd}

A handle to the window that owns the tray icon. This window will process any events generated by the icon.

\textit{uID}

The application-defined identifier that uniquely identifies the tray icon. This value allows a single window to own multiple tray icons, allowing the window to tell the difference between them.

\textit{uFlags}

A combination of the following flags specifying which of the subsequent members of the structure contain useful data. Any data members not identified by a flag will be ignored.

- \textbf{NIF_ICON} \quad The \textit{hIcon} data member.
- \textbf{NIF_MESSAGE} \quad The \textit{uCallbackMessage} data member.
- \textbf{NIF_TIP} \quad The \textit{szTip} data member.
- \textbf{NIF_STATE} \quad \textbf{Windows 2000:} The \textit{dwState} and \textit{dwStateMask} data members.
- \textbf{NIF_INFO} \quad \textbf{Windows 2000:} Use a balloon-style tooltip instead of the regular pop-up tooltip. The \textit{szInfo}, \textit{szTimeOut}, \textit{szInfoTitle}, and \textit{dwInfoFlags} data members are used.

\textit{uCallbackMessage}

An application-defined message identifier. This message is sent to the owning window whenever an event occurs related to the tray icon. These events are primarily when the mouse moves or clicks over the icon or when it receives keyboard input. The \textit{wParam} parameter will be the application-defined identifier of the tray icon that generated the message. The \textit{lParam} parameter will be the "real" Windows message identifier for the event that occurred.

\textit{hIcon}

A handle to the icon to display in the tray.

\textit{szTip}

The null-terminated string to use for the icon's standard tooltip text. This text appears when the mouse cursor hovers over the icon.

\textit{dwState}

\textbf{Windows 2000:} A combination of the following flags specifying the icon's state. Each flag is an independent toggle.

- \textbf{NIS_HIDDEN} \quad The icon is hidden.
- \textbf{NIS_SHAREDICON} \quad The icon is shared.
Windows API Guide: NOTIFYICONDATA Structure

**dwStateMask**

**Windows 2000:** A combination of the above flags that specifies which of the *dwState* flags to retrieve or modify.

**szInfo**

**Windows 2000:** The null-terminated string to use for the icon's balloon-style tooltip.

**uTimeoutOrVersion**

**Windows 2000:** The timeout value, in milliseconds, for the display of the balloon-style tooltip. If outside of Windows's allowable range, the timeout value will be "pushed" into the range of valid timeout periods.

Or, if [Shell_NotifyIcon](http://216.26.168.92/vbapi/ref/n/shell_notifyicon.html) had been called with the NIM_VERSION flag, this instead specifies whether to use Windows 95 or Windows 2000 behavior. By default, Windows 2000 will use different messages in some cases to communicate with the owning window. For backward compatibility, you should use the older behavior. This is one of the following flags:

- `0` Use the Windows 95-style behavior.
- `NOTIFYICON_VERSION` Use the Windows 2000-style behavior.

**szInfoTitle**

The null-terminated string to use as the title of the balloon tooltip. This will appear in boldface above the regular text.

**dwInfoFlags**

If desired, one of the following flags specifying the icon to display to the left of the balloon tooltip text. Set this as 0 to not use an icon.

- `NIIF_WARNING` A warning icon.
- `NIIF_ERROR` An error icon.
- `NIIF_INFO` An information icon.

**Constant Definitions**

```
Const NIF_ICON = &H2
Const NIF_MESSAGE = &H1
Const NIF_TIP = &H4
Const NIF_STATE = &H8
Const NIF_INFO = &H10
Const NIS_HIDDEN = &H1
Const NIS_SHAREDICON = &H2
Const NOTIFYICON_VERSION = &H1
Const NIIF_WARNING = &H30
```
Const NIIF_ERROR = &H10
Const NIIF_INFO = &H40

Used By

Shell_NotifyIcon

Back to the Structure list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/n/notifyicondata.html
NUMBERFMT Structure

Type NUMBERFMT
    NumDigits As Long
    LeadingZero As Long
    Grouping As Long
    lpDecimalSep As String
    lpThousandSep As String
    NegativeOrder As Long
End Type

Description & Usage

The NUMBERFMT structure stores information about how to format a regular number for display. This structure allows a program to specify how it wants a number to be displayed, overriding the format used by a locale.

Visual Basic-Specific Issues

None.

Data Members

NumDigits
    The number of digits to display after the decimal point.

LeadingZero
    If zero, do not pad the space to the right of the decimal point with zeros if there are fewer fractional digits than specified by NumDigits. If nonzero, then do pad the space. For example, if NumDigits is 3 and the number to display is 1.23, setting this data member to zero displays the number as "1.23". Setting this data member to any other value displays "1.230".

Grouping
    The number of digits to include in each group to the left of the decimal point. Typically, groups of three are used (e.g., 1,234,567). Values in the range of 0-9 are valid.
lpDecimalSep
   The character to use for the decimal point.

lpThousandSep
   The character to use for the grouping separator.

NegativeOrder
   One of the following values specifying how to represent a negative number. An example of each is shown in the list below for the number -1.1.
   0
       (1.1)
   1
       -1.1
   2
       - 1.1 (space after the - sign)
   3
       1.1-
   4
       1.1 - (space before the - sign)

Used By

GetNumberFormat

Back to the Structure list.
Back to the Reference section.

__________________________________________________________

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/n/numberfmt.html
OPENFILENAME Structure

Type OPENFILENAME

    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustomFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String

End Type

Description & Usage

The OPENFILENAME structure is used to pass data to and from the GetOpenFileName and GetSaveFileName API functions. It stores both settings used to create the dialog box and the results of the user's selection.

Visual Basic-Specific Issues

Windows NT, 2000: If you do not use a string in the structure, it is imperative that you not set it to an
empty string ("""). If you do not use a string, leave it alone entirely. Any strings you do use must be explicitly terminated by a null character (vbNullChar). Failure to do either of these will cause GetOpenFileName and GetSaveFileName to fail. See the examples for those two functions for a demonstration on how strings must be handled. Of course, following these guidelines still works under Windows 95 and 98, but it isn't necessary. Your best bet is to follow the NT/2000 requirements so your code works under all versions of Windows.

**Data Members**

$lStructSize

The size in bytes of the structure.

$hwndOwner

A handle to the window opening the file dialog box.

$hInstance

If using a dialog box template, this is a handle to the memory block of the dialog box template to use. If using the default dialog box, set to 0.

$lpstrFilter

The entries in the File Type drop box. The format of the string is "name of file type" & vbNullChar & "mask" & vbNullChar ... for as many types, where name of file type is the text that appears in the list and mask is the extension mask. The string must end with a double vbNullChar.

$lpstrCustomFilter

Similar to $lpstrFilter, but holds only one file type name/mask pair that specifies a user-defined file type. If unused, set to an empty string.

$nMaxCustFilter

The size in bytes of the string contained in $lpstrCustomFilter.

$nFilterIndex

The number (#1, #2, etc.) of data type specified $lpstrFilter should be the default one.

$lpstrFile

Set it as a series of blank spaces. If you want to specify a default filename, set this string to that default filename and pad it with spaces. Receives the complete path and filename of the file(s) the user selects. If multiple files are selected, each filename is separated by vbNullChar, and the entire string will end with a double vbNullChar.

$nMaxFile

The length in characters of $lpstrFile.

$lpstrFileTitle

Very similar to $lpstrFile, but only receives the filename of the selected file. If multiple files are selected, this is not set to any useful data.

$nMaxFileTitle

The length in characters of $lpstrFileTitle.

$lpstrInitialDir

The default directory to look in.

$lpstTitle
The text that appears in the dialog box's title bar.

**flags**

Zero or more of the following flags specifying how to create the file dialog box. Some of these flags will be set by the function after the call to reflect the user's selections.

**OFN_ALLOWMULTISELECT**
- Allow the user to select multiple files (Open File dialog box only).

**OFN_CREATEPROMPT**
- Prompt if a non-existing file is chosen.

**OFN_ENABLEHOOK**
- Use the function specified by `lpfnHook` to process the dialog box's messages.

**OFN_ENABLESIZING**
- **Windows 98, 2000:** Allow the dialog box to be resized. This is selected by default unless a hook function or custom template is specified.

**OFN_ENABLETEMPLATE**
- Use the dialog box template specified by `hInstance` and `lpTemplateName`.

**OFN_ENABLETEMPLATEHANDLE**
- Use the preloaded dialog box template specified by `hInstance`.

**OFN_EXPLORER**
- Use Windows Explorer-like additions to the file dialog box. This is selected by default unless a hook function or custom template is specified.

**OFN_EXTENSIONDIFFERENT**
- The function sets this flag if the user selects a file with an extension different than the one specified by `lpstrDefExt`.

**OFN_FILEMUSTEXIST**
- Only allow the selection of existing files.

**OFN_HIDEREADONLY**
- Hide the Open As Read Only check box (Open File dialog box only).

**OFN_LONGNAMES**
- Have the file dialog use long file names. This is automatically specified unless the Explorer-type extensions are not used.

**OFN_NOCHANGEDIR**
- Don't change Windows's current directory to match the one chosen in the dialog box.

**OFN_NODEREFERENCING**
- If a shortcut file (.lnk or .pif) is chosen, return the shortcut file itself instead of the file or directory it points to.

**OFN_NOLONGNAMES**
- Have the file dialog use short (8.3) file names. This is ignored unless a file dialog without Explorer-type extensions are not used.

**OFN_NONETWORKBUTTON**
- Hide and disable the Network button in the dialog box.

**OFN_NOREADONLYRETURN**
- The function sets this flag if the selected file is not read-only (Open File dialog box only).

**OFN_NOTESTFILECREATE**
Do not create a test file before the box closes. Normally, this check is done to verify that the disk exists, that there is sufficient disk space, etc. However, this check should not be used on a create-nonmodify network share. Specifying this flag prevents this test from being done.

**OFN_NOVALIDATE**
Don't check the filename for invalid characters.

**OFN_OVERWRITEPROMPT**
Prompt the user if the chosen file already exists (Save File dialog box only).

**OFN_PATHMUSTEXIST**
Only allow the selection of existing paths.

**OFN_READONLY**
Check the Open As Read Only box. This flag is set after the function call if the box is checked after the user clicks OK.

**OFN_SHAREAWARE**
Ignore any file sharing violations.

**OFN_SHOWHELP**
Show the Help button in the dialog box. The button sends the `WM_HELP` message to the hook function specified by the structure. If no hook function is used, the Help button will do nothing.

**nFileOffset**
Receives the zero-based index specifying where in `lpszFile` the pathname ends and the filename begins.

**nFileExtension**
Receives the zero-based index specifying where in `lpszFile` the file extension begins.

**lpszDefExt**
The default extension of a file (only for the Save dialog box). If a file is chosen with the `*.*` mask, the file gets this extension. Don't include the period.

**lCustData**
Information to pass to the hook function specified by `lPFNHook` whenever it is called.

**lPFNHook**
A pointer to a hook function to use to processes the dialog box's messages. For file dialogs with Explorer-type extensions, this is a pointer to a `OFNHookProc` hook function. For file dialogs without those extensions, this is a pointer to a `OFNHookProcOldStyle` hook function. If not using a hook function, set to 0.

**lpTemplateName**
The name of the dialog box template specified by `hInstance`, if needed.

## Constant Definitions

```plaintext
Const OFN_ALLOWMULTISELECT = &H200
Const OFN_CREATEPROMPT = &H2000
Const OFN_ENABLEHOOK = &H20
```
Const OFN_ENABLESIZING = &H800000
Const OFN_ENABLETEMPLATE = &H40
Const OFN_ENABLETEMPLATEHANDLE = &H80
Const OFN_EXPLORER = &H80000
Const OFN_EXTENSIONDIFFERENT = &H400
Const OFN_FILEMUSTEXIST = &H1000
Const OFN_HIDEREADONLY = &H4
Const OFN_LONGNAMES = &H200000
Const OFN_NOCHANGEDIR = &H8
Const OFN_NODEREFERENCIELINKS = &H100000
Const OFN_NOLONGNAMES = &H40000
Const OFN_NONETWORKBUTTON = &H20000
Const OFN_NOREADONLYRETURN = &H8000
Const OFN_NOTESTFILECREATE = &H10000
Const OFN_NOVALIDATE = &H100
Const OFN_OVERWRITEPROMPT = &H2
Const OFN_PATHMUSTEXIST = &H800
Const OFN_READONLY = &H1
Const OFN_SHAREAWARE = &H4000
Const OFN_SHOWHELP = &H10

**Used By**

GetOpenFileName, GetSaveFileName

Go back to the Structure listing.
Go back to the Reference section index.
OSVERSIONINFO Structure

Type OSVERSIONINFO
    dwOSVersionInfoSize As Long
    dwMajorVersion As Long
    dwMinorVersion As Long
    dwBuildNumber As Long
    dwPlatformId As Long
    szCSDVersion As String * 128
End Type

OSVERSIONINFO-type variables hold information about the version of Windows currently running. This structure holds various pieces of information identifying the version number, platform, and more about Windows.

- **dwOSVersionInfoSize**: The size of the structure.
- **dwMajorVersion**: The major version number; i.e., the part of the version number before the first period.
- **dwMinorVersion**: The minor version number; i.e., the part of the version number after the first period.
- **dwBuildNumber**: The build number of the version.
- **dwPlatformID**: Exactly one of the following flags identifying which platform of Windows is running (for example, Windows 95, Windows NT, etc.):
  - VER_PLATFORM_WIN32s = 0
    Windows 3.x is running, using the Win32s pseudo-32-bit enhancements.
  - VER_PLATFORM_WIN32_WINDOWS = 1
    Windows 95 or 98 is running.
  - VER_PLATFORM_WIN32_NT = 2
    Windows NT is running.
- **szCSDVersion**: More information about the operating system.

Used by: GetVersionEx
OVERLAPPED Structure

Type OVERLAPPED
    Internal As Long
    InternalHigh As Long
    Offset As Long
    OffsetHigh As Long
    hEvent As Long
End Type

Description & Usage

The OVERLAPPED structure tells file access functions what part of a file to read or write when asynchronous (overlapped) file access is used. The structure mainly holds the location within the file to begin the read or write operation.

Visual Basic-Specific Issues

None.

Data Members

Internal
    Used by Windows -- do not set.
InternalHigh
    Used by Windows -- do not set.
Offset
    Low-order dword of the 64-bit position in the file to start reading or writing to.
OffsetHigh
    High-order dword of the 64-bit position in the file to start reading or writing to.
hEvent
    Receives a handle to an event identifying the signaled state once the transfer is complete.
Used By

ReadFile, WriteFile

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: November 17, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/o/overlapped.html
POINT_TYPE Structure

Type POINT_TYPE
  x As Long
  y As Long
End Type

Description & Usage

The POINT_TYPE structure holds the (x,y) coordinate of a point. This structure is used throughout the API for storing the coordinates of a point.

Visual Basic-Specific Issues

Officially, this structure is called POINT. However, that violates the case-sensitive name spacing of Visual Basic because Visual Basic contains an intrinsic function called Point. The Windows API Guide calls this structure POINT_TYPE to avoid the naming collision. (Microsoft instead likes to call this structure POINT_API in Visual Basic contexts.)

Data Members

x
  The x coordinate of the point.

y
  The y coordinate of the point.

Used By

CreatePolygonRgn, CreatePolyPolygonRgn, GetBrushOrgEx, GetCursorPos, HELPINFO, LOGPEN, MAKEPOINTS, MoveToEx, PolyBezier, PolyBezierTo, Polygon, Polyline, PolyPolygon, PolylineTo, PolyPolyline, SetBrushOrgEx, WinHelp
PRINTDLG_TYPE Structure

```vbnet
Type PRINTDLG_TYPE
    lStructSize As Long
    hwndOwner As Long
    hDevMode As Long
    hDevNames As Long
    hdc As Long
    flags As Long
    nFromPage As Integer
    nToPage As Integer
    nMinPage As Integer
    nMaxPage As Integer
    nCopies As Integer
    hInstance As Long
    lCustData As Long
    lpfnPrintHook As Long
    lpfnSetupHook As Long
    lpPrintTemplateName As String
    lpSetupTemplateName As String
    hPrintTemplate As Long
    hSetupTemplate As Long
End Type
```

PRINTDLG_TYPE-type variables store the necessary information to use a Print common dialog box or a Print Setup common dialog box. This structure holds all information necessary to initialize the box and receives the data selected by the user. Set the various members for the box's default selections -- they will be set by the called function to the selections of the user. Note that two other structures, DEVMODE and DEVNAMES, are included by specifying handles to a memory block. These memory blocks contain a copy of each structure's data. See the example for PrintDlg for information on how to create, use, and free these memory blocks.

`lStructSize`

The size in bytes of this structure.

`hwndOwner`

A handle to the window opening the dialog box, if any.
**hDevMode**
A handle to the memory block holding the information contained in a **DEVMODE** structure. This data specifies information about the printer.

**hDevNames**
A handle to the memory block holding the information contained in a **DEVNAMES** structure. This data specifies the driver name, printer name, and port name(s) of the printer.

**hdc**
Receives either a device context or an information context (depending on the value set as **flags**) to the printer the user selected.

**flags**
Zero or more of the following flags specifying various options for creating the Print or Print Setup dialog. Note that when **PrintDlg** returns, many of these flags will be set by the function to indicate selections by the user:

- **PD_ALLPAGES = **&H0
  Select the All Pages radio button.

- **PD_COLLATE = **&H10
  Check the Collate check box. If this flag is set when the function returns, the user checked the box and the printer doesn't automatically support collation. If the box is checked and the printer does support it, this flag will not be set.

- **PD_DISABLEPRINTTOFILE = **&H80000
  Disable the Print to File check box.

- **PD_ENABLEPRINTHOOK = **&H1000
  Use the hook function pointed to by **lpfnPrintHook** to process the Print dialog box's messages.

- **PD_ENABLEPRINTTEMPLATE = **&H4000
  Use the Print dialog box template specified by **lpPrintTemplateName**.

- **PD_ENABLEPRINTTEMPLATEHANDLE = **&H10000
  Use the preloaded Print dialog box template specified by **hPrintTemplate**.

- **PD_ENABLESETUPHOOK = **&H2000
  Use the hook function pointed to by **lpfnSetupHook** to process the Print Setup dialog box's messages.

- **PD_ENABLESETUPTEMPLATE = **&H8000
  Use the Print Setup dialog box template specified by **lpSetupTemplateName**.

- **PD_ENABLESETUPTEMPLATEHANDLE = **&H20000
  Use the preloaded Print Setup dialog box template specified by **hSetupTemplate**.

- **PD_HIDEPRINTTOFILE = **&H100000
  Hide the Print to File check box.

- **PD_NONETWORKBUTTON = **&H200000
  Do not display any buttons associated with the network.

- **PD_NOPAGENUMS = **&H8
  Disable the Page Range radio button and edit boxes.

- **PD_NOSELECTION = **&H4
  Disable the Selection radio button.
PD_NOWARNING = &H80
Do not warn the user if there is no default printer.

PD_PAGENUMS = &H2
Select the Page Range radio button.

PD_PRINTSETUP = &H40
Display the Print Setup dialog box instead of the Print dialog box.

PD_PRINTTOFILE = &H20
Select the Print to File check box.

PD_RETURNDSC = &H100
Return a device context to the selected printer as hdc.

PD_RETURNDEFAULT = &H400
Instead of displaying either dialog box, simply load information about the default printer into hDevMode and hDevNames. For this to work, those two values must be set to 0 before calling the function.

PD_RETURNIC = &H200
Return an information context to the selected printer as hdc.

PD_SELECTION = &H1
Select the Selection radio button.

PD_SHOWHELP = &H800
Display the Help button.

PD_USEDEVMODECOPIES = &H40000
Same as PD_USEDEVMODECOPIESANDCOLLATE.

PD_USEDEVMODECOPIESANDCOLLATE = &H40000
If the printer does not automatically support multiple copies or collation, disable the corresponding options in the dialog box. The number of copies to print and the collation setting will be placed into hDevMode. The information returned to this structure will specify the number of pages and the collation which the program must print with -- the printer will print the copies or collate itself.

**nFromPage**
The value entered in the From Page text box, specifying which page begin printing at.

**nToPage**
The value entered in the To Page text box, specifying which page to stop printing at.

**nMinPage**
The minimum allowable value for nFromPage and nToPage.

**nMaxPage**
The maximum allowable value for nFromPage and nToPage.

**nCopies**
The number of copies the program needs to print.

**hInstance**
A handle to the application instance which has the desired dialog box template.

**lCustData**
A program-defined value to pass to whichever hook function is used.

**lpfnPrintHook**
A handle to the program-defined hook function to use to process the Print dialog box's messages.

*lpfnSetupHook*
A handle to the program-defined hook function to use to process the Print Setup dialog box's messages.

*lpPrintTemplateName*
The name of the Print dialog box template to use from the application instance specified by *hInstance*.

*lpSetupTemplateName*
The name of the Print Setup dialog box template to use from the application instance specified by *hInstance*.

*hPrintTemplate*
A handle to the preloaded Print dialog box template to use.

*hSetupTemplate*
A handle to the preloaded Print Setup dialog box template to use.

**Used by:** PrintDlg
PRINTER_DEFAULTS Structure

```vbnet
Type PRINTER_DEFAULTS
    pDatatype As String
    pDevMode As DEVMODE
    DesiredAccess As Long
End Type
```

**Description & Usage**

The **PRINTER_DEFAULTS** structure specifies some settings for opening a printer. The structure also contains information to use to initialize the printer.

**Visual Basic-Specific Issues**

None.

**Data Members**

- **pDatatype**
  - The name of the default data type for the printer.

- **pDevMode**
  - Information used to provide the initialization settings for the printer.

- **DesiredAccess**
  - **Windows NT, 2000**: One of the following flags specifying access rights to the printer:
    - PRINTER_ACCESS_ADMINISTER
      - Access to perform administrative tasks.
    - PRINTER_ACCESS_USE
      - Access to perform basic printing operations.
    - PRINTER_ALL_ACCESS
      - Access to perform all administrative and basic printing tasks except for synchronization.
  - **Windows 95, 98**: Reserved -- set to 0.
Constant Definitions

Const PRINTER_ACCESS_ADMINISTRATOR = &H4
Const PRINTER_ACCESS_USE = &H8
Const PRINTER_ALL_ACCESS = &HF000C

Used By

OpenPrinter

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: November 28, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/p/printer_defaults.html
**PRINTER_INFO_1 Structure**

Type PRINTER_INFO_1
   flags As Long
   pDescription As String
   pName As String
   pComment As String
End Type

**PRINTER_INFO_1**-type variable store information about a printer. This structure stores a few pieces of information relating to a printer.

**flags**

One or more of the following flags providing information about the printer:

- **PRINTER_ENUM_CONTAINER = &H8000**
  
  The object the structure describes is a container, such as a print server controlling multiple printers.

- **PRINTER_ENUM_EXPAND = &H4000**
  
  The program should further enumerate the printer if default expansion on it is enabled.

- **PRINTER_ENUM_ICON1 = &H10000**
  
  To represent the printer, the program should use an icon representing a top-level network name.

- **PRINTER_ENUM_ICON2 = &H20000**
  
  To represent the printer, the program should use an icon representing a network domain.

- **PRINTER_ENUM_ICON3 = &H40000**
  
  To represent the printer, the program should use an icon representing a print server.

- **PRINTER_ENUM_ICON8 = &H800000**
  
  To represent the printer, the program should use an icon representing a printer.

**pDescription**

A description of the printer.

**pName**

The name of the printer.

**pComment**

Comments or a brief description of the printer.

**Used by:** [EnumPrinters](http://216.26.168.92/vbapi/ref/p/printer_info_1.html)
**PRINTER_INFO_2 Structure**

Type PRINTER_INFO_2

```
pServerName As String
pPrinterName As String
pShareName As String
pPortName As String
pDriverName As String
pComment As String
pLocation As String
pDevMode As DEVMODE
pSepFile As String
pPrintProcessor As String
pDatatype As String
pParameters As String
pSecurityDescriptor As SECURITY_DESCRIPTOR
Attributes As Long
Priority As Long
DefaultPriority As Long
StartTime As Long
UntilTime As Long
Status As Long
cJobs As Long
AveragePPM As Long
```
End Type

PRINTER_INFO_2-type variables hold a multitude of information about a printer. Its member values, along with substructures inside of it, identify most pieces of information about a printer which Windows can provide.

**pServerName**

The name of the network server which controls the printer, if any.

**pPrinterName**

The name of the printer.

**pShareName**

The name of the sharepoint of the printer on the network, if any.
pPortName
   A comma-separated list of the printer port(s) the printer is connected to, such as LPT1:.

pDriverName
   The name of the printer driver.

pComment
   A comment about or a brief description of the printer.

pLocation
   The physical location of the printer (usually applies to network printers).

pDevMode
   Various default settings and attributes of the printer.

pSepFile
   The file that contains the separator page printed between jobs.

pPrintProcessor
   The name of the print processor the printer uses.

pDataType
   The name of the data type used to record the print jobs.

pParameters
   Default parameters for the print processor.

pSecurityDescriptor
   Security information about the printer.

Attributes
   One or more of the following flags specifying various attributes of the printer:
   PRINTER_ATTRIBUTE_DEFAULT = &H4
      The default printer.
   PRINTER_ATTRIBUTE_DIRECT = &H2
      There is a direct connection to the printer (?).
   PRINTER_ATTRIBUTE_DO_COMPLETE_FIRST = &H200
      Complete jobs on a first-come, first-serve basis (?).
   PRINTER_ATTRIBUTE_ENABLE_BIDI = &H800
      Win 95/98 only: BIDI is enabled (?).
   PRINTER_ATTRIBUTE_ENABLE_DEVQ = &H80
      DEVQ is enabled (?).
   PRINTER_ATTRIBUTE_KEEPPRINTEDJOBS = &H100
      The Printer keeps information on printed jobs (?).
   PRINTER_ATTRIBUTE_QUEUED = &H1
      The printer supports document queueing (?).
   PRINTER_ATTRIBUTE_SHARED = &H8
      The printer is shared on a network.
   PRINTER_ATTRIBUTE_WORK_OFFLINE = &H400
      Win 95/98 only: The printer can work offline (?).

Priority
   The priority given to the printer by the print spooler.

DefaultPriority
The default priority for a print job.

**StartTime**

The earliest time the printer will print a job, specified in minutes after midnight UTC (GMT or Zulu time).

**UntilTime**

The latest time the printer will print a job, specified in minutes after midnight UTC (GMT or Zulu time).

**Status**

One or more of the following flags specifying the printer's current status (Win NT only supports the PRINTER_STATUS_PAUSED and PRINTER_STATUS_PENDING_DELETION flags):

- **PRINTER_STATUS_BUSY** = &H200
  - The printer is busy.
- **PRINTER_STATUS_DOOR_OPEN** = &H400000
  - The door on the printer is open.
- **PRINTER_STATUS_ERROR** = &H2
  - An error has occurred.
- **PRINTER_STATUS_INITIALIZING** = &H8000
  - The printer is initializing.
- **PRINTER_STATUS_IO_ACTIVE** = &H100
  - I/O with the printer is active.
- **PRINTER_STATUS_MANUAL_FEED** = &H20
  - The printer is loading paper using manual feed.
- **PRINTER_STATUS_NO_TONER** = &H40000
  - The printer is out of toner.
- **PRINTER_STATUS_NOT_AVAILABLE** = &H1000
  - The printer is not available.
- **PRINTER_STATUS_OFFLINE** = &H80
  - The printer is offline.
- **PRINTER_STATUS_OUT_OF_MEMORY** = &H200000
  - The printer is out of memory.
- **PRINTER_STATUS_OUTPUT_BIN_FULL** = &H800
  - The printer's output bin is full.
- **PRINTER_STATUS_PAGE_PUNT** = &H80000
  - The printer has aborted printing the current page because it is too complex to handle.
- **PRINTER_STATUS_PAPER_JAM** = &H8
  - The printer's paper has jammed.
- **PRINTER_STATUS_PAPER_OUT** = &H10
  - The printer is out of paper.
- **PRINTER_STATUS_PAPER_PROBLEM** = &H40
  - There is a problem with the paper in the printer.
- **PRINTER_STATUS_PAUSED** = &H1
  - The printer is paused.
- **PRINTER_STATUS_PENDING_DELETION** = &H4
A document in the print queue is pending deletion.

PRINTER_STATUS_PRINTING = &H400
The printer is printing.

PRINTER_STATUS_PROCESSING = &H4000
The printer is processing information.

PRINTER_STATUS_TONER_LOW = &H20000
The printer is low on toner.

PRINTER_STATUS_USER_INTERVENTION = &H100000
The user has intervened in printer operations.

PRINTER_STATUS_WAITING = &H2000
The printer is waiting.

PRINTER_STATUS_WARMING_UP = &H10000
The printer is warming up.

AveragePPM
The average number of pages the printer can print per minute.

**Used by:** [EnumPrinters](http://www.vbapi.com/ref/p/printer_info_2.html)
PRINTER_INFO_4 Structure

Type PRINTER_INFO_4
  pPrinterName As String
  pServerName As String
  Attributes As Long
End Type

PRINTER_INFO_4-type variables store a very terse set of information about a printer. The information is limited to the name of the printer and on which network server (if any) it is located on.

pPrinterName
  The name of the printer.

pServerName
  The name of the network server the printer is on, if it is a network printer.

Attributes
  Exactly one of the following flags specifying whether the printer is locally connected or is on the network:
  PRINTER_ATTRIBUTE_LOCAL = &H40
      The printer is located on the network.
  PRINTER_ATTRIBUTE_NETWORK = &H10
      The printer is directly connected to the computer.

Used by: EnumPrinters

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
PRINTER_INFO_5 Structure

Type PRINTER_INFO_5
   pPrinterName As String
   pPortName As String
   Attributes As Long
   DeviceNotSelectedTimeout As Long
   TransmissionRetryTimeout As Long
End Type

PRINTER_INFO_5-type variables hold information about a printer. This structure only identifies a few of the possible pieces of information associated with a printer.

pPrinterName
   The name of the printer.

pPortName
   A comma-separated list of the ports the printer is connected to, such as LPT1:.

Attributes
   Zero or more of the following flags identifying various attributes of the printer:
   PRINTER_ATTRIBUTE_DEFAULT = &H4
      The printer is the default printer.
   PRINTER_ATTRIBUTE_DIRECT = &H2
      The printer is physically connected to the computer (i.e., it is not a network printer).
   PRINTER_ATTRIBUTE_QUEUED = &H1
      The printer supports queueing (?).
   PRINTER_ATTRIBUTE_SHARED = &H8
      The printer is a network printer.
   PRINTER_ATTRIBUTE_WORK_OFFLINE = &H400
      The printer can work if the computer is not connected to the network (?).

DeviceNotSelectedTimeout
   The maximum time, in milliseconds, between attempts to select the printer.

TransmissionRetryTimeout
   The maximum time, in milliseconds, between document transmission retries.

Used by: EnumPrinters
PROCESSENTRY32 Structure

Type PROCESSENTRY32
   dwSize As Long
   cntUsage As Long
   th32ProcessID As Long
   th32DefaultHeapID As Long
   th32ModuleID As Long
   cntThreads As Long
   th32ParentProcessID As Long
   pcPriClassBase As Long
   dwFlags As Long
   szExeFile As String * 260
End Type

Description & Usage

The PROCESSENTRY32 structure holds information about a process. This information is retrieved from a system snapshot via either Process32First or Process32Next.

Visual Basic-Specific Issues

None.

Parameters

* dwSize
  The length in bytes of the structure.
* cntUsage
  The number of references to the process. The process terminates once this drops to zero.
* th32ProcessID
  Identifier of the process. This value can be used by other API functions that work with processes.
* th32DefaultHeapID
Identifier of the default heap for the process. This value can only be used by the tool help API functions and no others.

thModuleID
Identifier of the process's module. This value can only be used by the tool help API functions and no others.

cntThreads
The number of threads started by the program.

th32ParentProcessID
The identifier of the process that created this process. This value can be used by other API functions that work with processes.

pcPriClassBase
The base priority by any threads created by this class.

dwFlags
Reserved -- do not use.

szExeFile
The filename of the executable file for the process.

Used By

Process32First, Process32Next
RECT Structure

Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type

Description & Usage

The RECT structure holds a rectangle. This structure defines a rectangle by storing the coordinates of its upper-left and lower-right corners. Generally, points lying along the bottom or right edges of the rectangle are not considered to be inside the rectangle; however, points along the top or left edges are.

Visual Basic-Specific Issues

None.

Data Members

left
    The x-coordinate of the upper-left corner of the rectangle.

top
    The y-coordinate of the upper-left corner of the rectangle.

right
    The x-coordinate of the lower-right corner of the rectangle.

bottom
    The y-coordinate of the lower-right corner of the rectangle.

Used By
SECURITY_ATTRIBUTES Structure

Type SECURITY_ATTRIBUTES
    nLength As Long
    lpSecurityDescriptor As Long
    bInheritHandle As Boolean
End Type

Description & Usage

The SECURITY_ATTRIBUTES structure defines the level and type of security protection to give an object. Note that security information is used much more frequently under Windows NT/2000 than it is under Windows 95/98.

Visual Basic-Specific Issues

None.

Data Members

nLength
    The size in bytes of the structure.
lpSecurityDescriptor
    A pointer to a security descriptor for the object. To use the default security descriptor, set this to 0.
bInheritHandle
    Specifies whether the object's handle should be inherited by new processes or programs.

Used By

CreateDirectory, CreateDirectoryEx, CreateFile, RegCreateKeyEx
SECURITY_DESCRIPTOR

Type SECURITY_DESCRIPTOR
   Revision As Byte
   Sbz1 As Byte
   Control As Long
   Owner As Long
   Group As Long
   Sacl As ACL
   Dacl As ACL
End Type

Description & Usage

The SECURITY_DESCRIPTOR structure identifies security information about an object. This structure should NEVER be written to directly! Instead, use the set of API functions that read and write information to this structure.

Visual Basic-Specific Information

None.

Data Members

Revision
   Revision number of the information.
Sbz1
   Reserved -- set to 0. This member simply aligns the other members in memory.
Control
   Control identifier.
Owner
   Owner identifier.
Group
Group identifier.

*Sacl*
The system access-control list (ACL).

*Dacl*
The discretionary access-control list (ACL).

**Used By**

JOB_INFO_2, PRINTER_INFO_2

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** November 26, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000 Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/security_descriptor.html
SERIALKEYS Structure

**Type** SERIALKEYS

```
    cbSize As Long
    dwFlags As Long
    lpszActivePort As String
    lpszPort As String
    iBaudRate As Long
    iPortState As Long
End Type
```

**SERIALKEYS**-type variables store information about the SerialKeys accessibility feature. SerialKeys allows a device connected to a serial port to imitate the mouse and/or keyboard. This structure stores the settings for SerialKeys.

**cbSize**

The size in bytes of the structure.

**dwFlags**

Zero or more of the following flags specifying various settings and properties of SerialKeys:

- `SERKF_ACTIVE` = &H8
  SerialKeys is currently receiving input on the serial port.
- `SERKF_AVAILABLE` = &H2
  The SerialKeys feature is available.
- `SERKF_INDICATOR` = &H4
  Display an icon in the system tray while SerialKeys is on.
- `SERKF_SERIALKEYSON` = &H1
  SerialKeys is currently on.

**lpszActivePort**

The name of the serial port to read from. An empty string signifies no port. "Auto" means SerialKeys will monitor any otherwise unused serial ports.

**lpszPort**

Reserved.

**iBaudRate**

The baud rate of the port identified by `lpszActivePort`. Valid baud rates are 110, 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 38400, 56000, 57600, 115200, 128000, and 256000.

**iPortState**
Identifies the state of the port identified by `lpszActivePort`. 0 means that all input is ignored. 1 means that the port is being monitored for SerialKeys input when not being used by other programs. 2 means that all input is sent to SerialKeys.

**Used by:** SystemParametersInfo

[Go back to the alphabetical Structure listing.](http://www.vbapi.com/ref/s/serialkeys.html)

[Go back to the Reference section index.](http://www.vbapi.com/ref/s/serialkeys.html)
SHELLEXECUTEINFO Structure

Type SHELLEXECUTEINFO
    cbSize As Long
    fMask As Long
    hwnd As Long
    lpVerb As String
    lpFile As String
    lpParameters As String
    lpDirectory As String
    nShow As Long
    hInstApp As Long
    lpIDList As Long
    lpClass As String
    hkeyClass As Long
    dwHotKey As Long
    hIcon As Long
    hProcess As Long
End Type

Description & Usage

The SHELLEXECUTEINFO structure holds the information passed to and from the ShellExecuteEx function. The structure stores the data telling the function what to do, and also receives data describing the function's result.

Visual Basic-Specific Issues

Data Members

cbSize
The size, in bytes, of the structure.
fMask
A combination of the following flags specifying which optional parts of the structure to use, and also specifying other options to use with the structure:

**SEE_MASK_CLASSKEY**
- Use the **hkeyClass** member.

**SEE_MASK_CLASSNAME**
- Use the **lpClass** member.

**SEE_MASK_CONNECTNETDRV**
- Validate the file share and connect it to a drive letter. **lpFile** is the Universal Naming Convention (UNC) pathname of a file on the network.

**SEE_MASK_DOENVSUBST**
- Expand any environment variables appearing inside **lpDirectory** or **lpFile**.

**SEE_MASK_FLAG_DDEWAIT**
- If **ShellExecuteEx** starts a DDE conversation, wait for that conversation to finish before the function returns.

**SEE_MASK_FLAG_NO_UI**
- Do not display an error dialog if an error occurs.

**SEE_MASK_HOTKEY**
- Use the **dwHotKey** member.

**SEE_MASK_ICON**
- Use the **hIcon** member.

**SEE_MASK_IDLIST**
- Use the **lpIDList** member.

**SEE_MASK_INVOKEIDLIST**
- Use the **lpIDList** member to invoke an application.

**SEE_MASK_NOCLOSEPROCESS**
- Use the **hProcess** member.

**hwnd**
- A **handle** to the window that is calling **ShellExecuteEx**, if any.

**lpVerb**
- A string that specifies the action to perform on the file specified by **lpFile**. This may be one of the following strings, although others are possible:
  - "explore"
    - If **lpFile** is a path name, open it in a Windows Explorer window.
  - "open"
    - Open **lpFile** using its associated program. Opening an executable file runs it. This is the default action if none is specified.
  - "print"
    - Print **lpFile** using its associated program.

**lpFile**
- The name of the file to open, print, execute, or whatever is specified by **lpVerb**.

**lpParameters**
- Additional parameters to use to perform the action. This would typically be additional command-line options to use, especially when running an executable file.
**lpDirectory**
The path name of the working directory to use. If this is not specified, the current directory is used.

**nShow**
One of the following flags specifying how to display any window opened as a result of the call to ShellExecuteEx:

- **SW_HIDE**
  Hide the window.
- **SW_MAXIMIZE**
  Maximize the window.
- **SW_MINIMIZE**
  Minimize the window.
- **SW_RESTORE**
  Restore the window (not maximized nor minimized).
- **SW_SHOW**
  Show the window.
- **SW_SHOWMAXIMIZED**
  Show the window maximized.
- **SW_SHOWMINIMIZED**
  Show the window minimized.
- **SW_SHOWMINNOACTIVE**
  Show the window minimized but do not activate it.
- **SW_SHOWNA**
  Show the window in its current state but do not activate it.
- **SW_SHOWNOACTIVATE**
  Show the window in its most recent size and position but do not activate it.
- **SW_SHOWNORMAL**
  Show the window and activate it (as usual).

**hInstApp**
If ShellExecuteEx is successful, this receives a handle to the instance of the application that was started. If the function failed, this receives one of the following flags identifying the error:

- **SE_ERR_ACCESSDENIED**
  Access was denied.
- **SE_ERR_ASSOCINCOMPLETE**
  File association information was incomplete.
- **SE_ERR_DDEBUSY**
  The DDE operation is busy.
- **SE_ERR_DDEFAIL**
  The DDE operation failed.
- **SE_ERR_DDETIMEOUT**
  The DDE operation timed out.
- **SE_ERR_DLLNOTFOUND**
  A required DLL was not found.
SE_ERR_FNF
    The file could not be found.
SE_ERR_NOASSOC
    There is no associated program to use to perform the action.
SE_ERR_OOM
    The computer is out of memory.
SE_ERR_PNF
    The path could not be found.
SE_ERR_SHARE
    A shared file could not be opened.

lpIDList
    A pointer to an ITEMIDLIST structure (PIDL) that identifies the file to execute. This member is only used if fMask contains SEE_MASK_IDLIST or SEE_MASK_INVOKEIDLIST.

lpClass
    The name of the file class or globally unique identifier (GUID) to use. This member is only used if fMask contains SEE_MASK_CLASSNAME.

hkeyClass
    A handle to the registry key to use for the file class. This member is only used if fMask contains SEE_MASK_CLASSKEY.

dwHotKey
    The hot key to associate with the application. The low-order word is the virtual-key code of the hot key, and the high-order word is a combination of the following flags specifying the modifier keys to use:
    HOTKEYF_ALT
        Use the Alt key as a modifier.
    HOTKEYF_CONTROL
        Use the Ctrl key as a modifier.
    HOTKEYF_EXT
        Use the extended key as a modifier.
    HOTKEYF_SHIFT
        Use the Shift key as a modifier.
    This member is only used if fMask contains SEE_MASK_HOTKEY.

hIcon
    A handle to the icon to use for the file class. This member is only used if fMask contains SEE_MASK_ICON.

hProcess
    Receives a handle to the process that was started by the call to ShellExecuteEx. This member is only used if fMask contains SEE_MASK_NOCLOSEPROCESS.

Constant Definitions

Const SEE_MASK_CLASSKEY = &H3
Const SEE_MASK_CLASSNAME = &H1
Const SEE_MASK_CONNECTNETDRV = &H80
Const SEE_MASK_DOENV SUBST = &H200
Const SEE_MASK_FLAG_DDEWAIT = &H100
Const SEE_MASK_FLAG_NO_UI = &H400
Const SEE_MASK_HOTKEY = &H20
Const SEE_MASK_ICON = &H10
Const SEE_MASK_IDLIST = &H4
Const SEE_MASK_INVOKEIDLIST = &HC
Const SEE_MASK_NOCLOSEPROCESS = &H40
Const SW_HIDE = 0
Const SW_MAXIMIZE = 3
Const SW_MINIMIZE = 6
Const SW_RESTORE = 9
Const SW_SHOW = 5
Const SW_SHOWMAXIMIZED = 3
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMINNOACTIVE = 7
Const SW_SHOWNA = 8
Const SW_SHOWNOACTIVATE = 4
Const SW_SHOWNORMAL = 1
Const SE_ERR_ACCESSDENIED = 5
Const SE_ERR_ASSOCINCOMPLETE = 27
Const SE_ERR_DDEBUSY = 30
Const SE_ERR_DDEFAIL = 29
Const SE_ERR_DDETIMEOUT = 28
Const SE_ERR_DLLNOTFOUND = 32
Const SE_ERR_FNF = 2
Const SE_ERR_NOASSOC = 31
Const SE_ERR_OOM = 8
Const SE_ERR_PNF = 3
Const SE_ERR_SHARE = 26
Const HOTKEYF_ALT = &H4
Const HOTKEYF_CONTROL = &H2
Const HOTKEYF_EXT = &H8
Const HOTKEYF_SHIFT = &H1

Used By

ShellExecuteEx

Back to the Structure list.
SHFILEINFO Structure

Type SHFILEINFO
    hIcon As Long
    iIcon As Long
    dwAttributes As Long
    szDisplayName As String * 260
    szTypeName As String * 80
End Type

Description & Usage

The SHFILEINFO structure holds information about a file system object pertaining to the system shell. The structure can hold such information as a handle to its icon and its display name in the shell. Depending on the flags used in the SHGetFileInfo function, not all of the structure's data members may necessarily hold useful information.

Visual Basic-Specific Issues

None.

Data Members

hIcon
A handle to the icon used to represent the file system object in the shell. Your program must destroy this icon via DestroyIcon after using it to save resources.

iIcon
The index of the file system's icon within the system image list.

dwAttributes
A combination of the following flags specifying the attributes of the file system object:
FILE_ATTRIBUTE_ARCHIVE
    An archive file (which most files are).
FILE_ATTRIBUTE_COMPRESSED
A file residing in a compressed drive or directory.

FILE_ATTRIBUTE_DIRECTORY
A directory instead of a file.

FILE_ATTRIBUTE_HIDDEN
A hidden file, not normally visible to the user.

FILE_ATTRIBUTE_NORMAL
An attribute-less file (cannot be combined with other attributes).

FILE_ATTRIBUTE_READONLY
A read-only file.

FILE_ATTRIBUTE_SYSTEM
A system file, used exclusively by the operating system.

szDisplayName
A null-terminated string specifying the name of the file system object as it appears in the system shell. For files, this will be the full path and filename.

szTypeName
A null-terminated string specifying the name of the type of the file system object.

Constant Definitions

Const FILE_ATTRIBUTE_ARCHIVE = &H20
Const FILE_ATTRIBUTE_COMPRESSED = &H800
Const FILE_ATTRIBUTE_DIRECTORY = &H10
Const FILE_ATTRIBUTE_HIDDEN = &H2
Const FILE_ATTRIBUTE_NORMAL = &H0
Const FILE_ATTRIBUTE_READONLY = &H1
Const FILE_ATTRIBUTE_SYSTEM = &H4

Used By

SHGetFileInfo
Windows API Guide: SHFILEINFO Structure

This page is at http://www.vbapi.com/ref/s/shfileinfo.html
SHFILEOPSTRUCT Structure

Type SHFILEOPSTRUCT
    hwnd As Long
    wFunc As Long
    pFrom As String
    pTo As String
    fFlags As Integer
    fAnyOperationsAborted As Long
    hNameMappings As Long
    lpszProgressTitle As String
End Type

Description & Usage

The SHFILEOPSTRUCT structure holds a description of a file operation to have the SHFileOperation function perform. The contents of the structure identify the operation itself (copy, move, delete, or rename) as well as the targeted files and any other options. In some cases, the structure also receives feedback information based on the actions of the operation.

Visual Basic-Specific Issues

Because of how Visual Basic byte-aligns structure contents, the SHFILEOPSTRUCT cannot be passed directly to SHFileOperation or any other API function. Instead, it must be copied to a byte array in such a way to correct for the improper byte alignment. Specifically, bytes 1 through 18 of the structure must immediately precede bytes 21 through 32. In other words, bytes 19 and 20 must be removed. (The given byte positions are given so that the first byte is byte #1.) The example code for SHFileOperation demonstrates the most straightforward way to correct for this problem.

Data Members

hwnd

A handle to the window that is performing the file operation.
wFunc

One of the following flags specifying which file operation to perform:

FO_COPY
Copy the files specified by pFrom to the location specified by pTo.

FO_DELETE
Delete the files specified by pFrom. (To send files to the Recycle Bin, use this file operation and include the FOF_ALLOWUNDO flag in fFlags.)

FO_MOVE
Move the files specified by pFrom to the location specified by pTo.

FO_RENAME
Rename the files specified by pFrom to the names specified by pTo.

pFrom
A list of one or more files that are the source of the file operation. If multiple files are specified, they must be separated by a null character. The entire string must be terminated by two null characters. Wildcard characters (* and ?) are allowed in the filenames.

pTo
A list of one or more paths or files that are the target of the file operation. If multiple files or paths are specified, they must be separated by a null character. The entire string must be terminated by two null characters. Wildcard characters (* and ?) are sometimes allowed, depending on the operation. Always specify the entire path and filename for any files or directories.

fFlags
A combination of the following flags specifying other options or settings:

FOF_ALLOWUNDO
Allow the user to Undo the file operation, if possible. All paths must be fully qualified for this flag to work.

FOF_FILESONLY
If a *.* wildcard file is specified in pFrom or pTo, only perform the file operation on the files (and not the directories).

FOF_MULTIDESTFILES
pTo specifies multiple destination files -- one for each source file in pFrom -- instead of a single target for all source files.

FOF_NOCONFIRMATION
Do not prompt the user with any confirmation dialogs, instead assuming a "Yes to All" response.

FOF_NOCONFIRMMKDIR
Do not confirm the creation of a new directory if the file operation requires one to be made.

FOF_NO_CONNECTED_ELEMENTS
Windows 2000: Do not move connected files as a group; instead, move only the specified files.

FOF_NOCOPYSECURITYATTRIBS
Windows NT 4.71 and later, 2000: Do not copy the security attributes of the files.

FOF_NOERRORUI

Do not display a message box if an error occurs.

**FOF_NORECURSION**

Only perform the file operation in the specified directory, not operating recursively into its subdirectories.

**FOF_RENAMEONCOLLISION**

If a file being moved, copied, or renamed conflicts with an existing file, give the file operated on a new name.

**FOF_SILENT**

Do not display a progress dialog box.

**FOF_SIMPLEPROGRESS**

Display a progress dialog box displaying the text specified by `lpszProgressTitle` but do not display the file names being processed.

**FOF_WANTMAPPINGHANDLE**

Create a `SHNAMEMAPPING` structure to identify any files that must be renamed as a result of the **FOF_RENAMEONCOLLISION** flag. A handle to this structure is put into `hNameMappings`.

**FOF_WANTNUKEWARNING**

*Windows 2000:* Warn the user if a file is being deleted rather than being sent to the Recycle Bin.

`fAnyOperationsAborted`

Receives a nonzero value if the user aborted any file operations before they were complete. This receives 0 if all operations completed without user intervention.

`hNameMappings`

If `fFlags` contains **FOF_WANTMAPPINGHANDLE**, this receives a handle to an array of `SHNAMEMAPPING` structures that identify which files were renamed. After use, this handle must be freed by the `SHFreeNameMappings` function. **NOTE:** I have not been able to figure out how to use the mapping handle placed in this data member. If anyone knows how, please e-mail me.

`lpszProgressTitle`

If `fFlags` contains the **FOF_SIMPLEPROGRESS** flag, this is the text to display in the progress dialog box. This string must be null-terminated.

### Constant Definitions

```
Const FO_COPY = &H2
Const FO_DELETE = &H3
Const FO_MOVE = &H1
Const FO_RENAME = &H4
Const FOF_ALLOWUNDO = &H40
Const FOF_FILESONLY = &H80
Const FOF_MULTIDESTFILES = &H1
Const FOF_NOCONFIRMATION = &H10
```
Const FOF_NOCONFIRMMDIR = &H200
Const FOF_NO_CONNECTED_ELEMENTS = &H1000
Const FOF_NOCOPYSECURITYATTRIBS = &H800
Const FOF_NOERRORUI = &H400
Const FOF_RENAMEONCOLLISION = &H8
Const FOF_SILENT = &H4
Const FOF_SIMPLEPROGRESS = &H100
Const FOF_WANTMAPPINGHANDLE = &H20
Const FOF_WANTNUKEWARNING = &H2000

Used By

SHFileOperation

Back to the Structure list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/s/shfileopstruct.html
SHITEMID Structure

Type SHITEMID
    cbSize As Integer
    abID As String * 256  ' can actually be any size, not necessarily 256
End Type

Description & Usage

The SHITEMID structure holds the data of an item identifier. Since the ITEMIDLIST structure (to which this structure belongs) is almost never used explicitly by a program, this structure also is usually not used directly.

Visual Basic-Specific Issues

None.

Data Members

cbSize
    The size in bytes of the structure.

abID
    The data of the item identifier. The length of this parameter can vary. The format of its contents is undocumented and cannot be accessed directly.

Used By

ITEMIDLIST

Go back to the alphabetical Structure listing.

Go back to the Reference section index.
Type SHNAMEMAPPING

    pszOldPath As String
    pszNewPath As String
    cchOldPath As Long
    cchNewPath As Long

End Type

Description & Usage

The SHNAMEMAPPING structure identifies which files were renamed automatically as a result of a file operation. Files will automatically be renamed if a naming collision occurs during a copy, move, or rename operation performed by the SHFileOperation function.

Visual Basic-Specific Issues

None.

Data Members

pszOldPath
    The old path of the file.
pszNewPath
    The new path of the file.
cchOldPath
    The number of characters in pszOldPath.
cchNewPath
    The number of characters in pszNewPath.

Used By
SHQUERYRINFO Structure

Type SHQUERYRINFO
   cbSize As Long
   i64Size As ULLARGE_INTEGER
   i64NumItems As ULLARGE_INTEGER
End Type

Description & Usage

The SHQUERYRINFO holds information about a Recycle Bin. This Recycle Bin could be specific to a particular drive, or it could be the systemwide Recycle Bin. The structure holds the number of items in the Recycle Bin as well as the total size of all items in it. Both values are stored as unsigned 64-bit integers.

Visual Basic-Specific Issues

None.

Data Members

*cbSize*  
The size in bytes of the structure.

*i64Size*  
The total size, in bytes, of all the items currently in the Recycle Bin.

*i64NumItems*  
The total number of items currently in the Recycle Bin.

Used By

SHQueryRecycleBin
SOCKADDR Structure

Type SOCKADDR
    sin_family As Integer
    sin_port As Integer
    sin_addr As Long
    sin_zero As String * 8
End Type

Description & Usage

The SOCKADDR structure stores information about a connection to make to a network host. For TCP/IP connections, this is essentially the IP address of the remote host and the port to connect to. Other base protocols may have different requirements for the data to load into the structure, although no matter what, the structure will always be 16 bytes long.

Visual Basic-Specific Issues

None.

Data Members

sin_family
    The address family of the protocol to use. This will match the protocol used by the socket making the connection.

sin_port
    The port number to connect to, in network byte order.

sin_addr
    The IP address of the network host to connect to, in network byte order.

sin_zero
    Padding data. If not using it, set all bytes to zero.
Used By

connect

Back to the Structure list.
Back to the Reference section.
SOUNDSENTRY Structure

Type SOUNDSENTRY
    cbSize As Long
    dwFlags As Long
    iFSTextEffect As Long
    iFSTextEffectMSec As Long
    iFSTextEffectColorBits As Long
    iFSGrafEffect As Long
    iFSGrafEffectMSec As Long
    iFSGrafEffectColor As Long
    iWindowsEffect As Long
    iWindowsEffectMSec As Long
    lpszWindowsEffectDLL As String
    iWindowsEffectOrdinal As Long
End Type

Description & Usage

The SOUNDSENTRY structure stores information about the SoundSentry accessibility feature. SoundSentry displays a visual cue whenever a sound is made. **Windows NT, 2000:** SoundSentry works for sounds played through both the internal speaker and multimedia devices. **Windows 95, 98:** SoundSentry only works for sounds played through the internal speaker.

Visual Basic-Specific Issues

None.

Data Members

*cbSize*

The size in bytes of the structure.

*dwFlags*
Zero or more of the following flags specifying various settings and properties of SoundSentry:

SSF_AVAILABLE
   The SoundSentry accessibility feature is available.

SSF_INDICATOR
   Display an icon in the system tray while SoundSentry is on.

SSF_SOUNDSENTRYON
   SoundSentry is currently on.

**iFSTextEffect**

**Windows 95, 98:** Exactly one of the following flags specifying the visual cue to use while the user is in a full-screen text window (such as a full-screen MS-DOS window). **Windows NT, 2000:** Reserved -- set to 0.

SSTF_BORDER
   Flash the screen border (the overscan area).

SSTF_CHARS
   Flash the characters in the corners of the screen.

SSTF_DISPLAY
   Flash the entire display.

SSTF_NONE
   Do not display a visual cue.

**iFSTextEffectMSec**

**Windows 95, 98:** The length of time in milliseconds to display the visual cue while in full-screen text mode. **Windows NT, 2000:** Reserved -- set to 0.

**iFSTextEffectColorBits**

**Windows 95, 98:** The RGB value of the color to use for the visual cue while in full-screen text mode. **Windows NT, 2000:** Reserved -- set to 0.

**iFSGrafEffect**

**Windows 95, 98:** Exactly one of the following flags specifying the visual cue to display while the user is in a full-screen graphics window. **Windows NT, 2000:** Reserved -- set to 0.

SSGF_DISPLAY
   Flash the entire display.

SSGF_NONE
   Do not display a visual cue.

**iFSGrafEffectMSec**

**Windows 95, 98:** The length of time in milliseconds to display the visual cue while in full-screen graphics mode. **Windows NT, 2000:** Reserved -- set to 0.

**iFSGrafEffectColor**

**Windows 95, 98:** The RGB value of the color to use for the visual cue while in full-screen graphics mode. **Windows NT:** Reserved -- set to 0.

**iWindowsEffect**

Exactly one of the following flags specifying the visual cue to use while not in full-screen mode:

SSWF_CUSTOM
   Windows 95, 98: Call the SoundSentryProc callback function exported by the .dll file specified by lpszWindowsEffectDLL.
SSWF_DISPLAY
  Flash the entire display.
SSWF_NONE
  Do not display a visual cue.
SSWFITLE
  Flash the title bar of the active window.
SSWF_WINDOW
  Flash the entire active window.

\texttt{iWindowsEffectMSec}

\textbf{Windows 95, 98}: The length of time in milliseconds to display the visual cue while not in full-screen mode. \textbf{Windows NT, 2000}: Reserved -- set to 0.

\texttt{lpszWindowsEffectDLL}

\textbf{Windows 95, 98}: The filename of the .dll file exporting the \texttt{SoundSentryProc} callback function to use, if applicable. \textbf{Windows NT, 2000}: Reserved.

\texttt{iWindowsEffectOrdinal}
  Reserved -- set to 0.

\section*{Constant Definitions}

\begin{verbatim}
Const SSF_AVAILABLE = &H2
Const SSF_INDICATOR = &H4
Const SSF_SOUNDSENTRYON = &H1
Const SSTF_BORDER = 2
Const SSTF_CHARS = 1
Const SSTF_DISPLAY = 3
Const SSTF_NONE = 0
Const SSGF_DISPLAY = 3
Const SSGF_NONE = 0
Const SSWF_CUSTOM = 4
Const SSWF_DISPLAY = 3
Const SSWF_NONE = 0
Const SSWF_TITLE = 1
Const SSWF_WINDOW = 2
\end{verbatim}

\section*{Used By}

\texttt{SystemParametersInfo}

\textbf{Go back to the alphabetical Structure listing.}
\textbf{Go back to the Reference section index.}
STICKYKEYS Structure

Type STICKYKEYS
    cbSize As Long
    dwFlags As Long
End Type

STICKYKEYS-type variables store information about the StickyKeys accessibility feature. StickyKeys allows the user to simply press modifier keys (Ctrl, Alt, Shift) before pressing another key instead of having to press them simultaneously. For example, to give a Ctrl-S command, the user would simply press and release Ctrl, then press S (instead of pressing S while holding Ctrl). This structure holds the settings of StickyKeys.

cbSize
    The size in bytes of the structure.

dwFlags
    Zero or more of the following flags specifying various settings and properties of StickyKeys:
    SKF_AUDIBLEFEEDBACK = &H40
        Play a sound whenever the user latches, locks, or releases a modifier key.
    SKF_AVAILABLE = &H2
        The StickyKeys feature is available.
    SKF_CONFIRMHOTKEY = &H8
        Win 95/98 only: Open a confirmation dialog box when the user activates StickyKeys via the hot key.
    SKF_HOTKEYACTIVE = &H4
        Enable the user to toggle StickyKeys via the hot key: pressing Shift five times.
    SKF_HOTKEYSOUND = &H10
        Play a sound when the user toggles StickyKeys via the hot key.
    SKF_INDICATOR = &H20
        Win 95/98: Display an icon in the system tray while StickyKeys is on.
    SKF_STICKYKEYSON = &H1
        StickyKeys is currently on.
    SKF_TRISTATE = &H80
        Allow the user to press a modifier key two times in a row to lock it; it will be unlocked after a third press.
    SKF_TWOKEYSOFF = &H100
Holding a modifier key and a normal key together deactivates StickyKeys.

**Used by:** SystemParametersInfo

**Go back to the alphabetical Structure listing.**
**Go back to the Reference section index.**
SYSTEMTIME Structure

Type SYSTEMTIME  
    wYear As Integer  
    wMonth As Integer  
    wDayOfWeek As Integer  
    wDay As Integer  
    wHour As Integer  
    wMinute As Integer  
    wSecond As Integer  
    wMilliseconds As Integer  
End Type

Description & Usage

The **SYSTEMTIME** structure holds a date and time in an easily usable format. The structure is able to hold times precise to the millisecond, even though that information may not always be used.

Sometimes, the **SYSTEMTIME** structure is used not to hold a "regular" date and time but to hold a shifting one (such as the 1\textsuperscript{st} Sunday in January or the 3\textsuperscript{rd} Thursday of June). When this is used (usually when holding information relating to a time zone), \textit{wYear} is always 0. \textit{wDay} specifies which day, ranging from 1 (1\textsuperscript{st} day) to 5 (5\textsuperscript{th} or last day). \textit{wDayOfWeek} identifies the precise day of the week.

Visual Basic-Specific Issues

None.

Data Members

\textbf{wYear}  
   The four-digit year.

\textbf{wMonth}  
   The number identifying the month (1 = January, 2 = February, etc.).
**wDayOfWeek**
The number identifying the day of the week (0 = Sunday, 1 = Monday, etc.).

**wDay**
The number of the day of the month.

**wHour**
The hour, in twenty-four hour format.

**wMinute**
The minutes.

**wSecond**
The seconds.

**wMilliseconds**
The milliseconds.

---

**Used By:**

- FileTimeToSystemTime
- GetDateFormat
- GetLocalTime
- GetSystemTime
- GetTimeFormat
- JOB_INFO_1
- JOB_INFO_2
- SetSystemTime
- SystemTimeToFileTime
- TIME_ZONE_INFORMATION

---

**Last Modified:** January 3, 2000
This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/s/systemtime.html) Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: [vbapi@vbapi.com](mailto:vbapi@vbapi.com) Send Encrypted E-Mail
This page is at [http://www.vbapi.com/ref/s/systemtime.html](http://www.vbapi.com/ref/s/systemtime.html)
TEXTMETRIC Structure

The **TEXTMETRIC** structure holds the text metrics of a font. The structure describes many physical attributes of the font. Unless otherwise specified, all size measurements of the font stored in the structure are in the logical units of the device the font is being used on, so the same font may have a different "scale" if used on a different device.

**Visual Basic-Specific Issues**
Data Members

\textit{tmHeight}  
The height of the font's characters (equal to the sum of \textit{tmAscent} and \textit{tmDescent}).

\textit{tmAscent}  
The ascent (units above the base line) of the font's characters.

\textit{tmDescent}  
The descent (units below the base line) of the font's characters.

\textit{tmInternalLeading}  
The amount of leading space inside the bounds of \textit{tmHeight} where accent marks or other diacritical marks may appear.

\textit{tmExternalLeading}  
The amount of leading space placed between rows of text in which the font does not draw.

\textit{tmAveCharWidth}  
The average width of the font's characters, usually identified as the width of the "x" character. This value does not include extra space required for bold or italic characters.

\textit{tmMaxCharWidth}  
The width of the widest character of the font. This value does not include extra space required for bold or italics.

\textit{tmWeight}  
One of the following flags specifying the boldness (weight) of the font:
\begin{itemize}
  \item \texttt{FW\_DONTCARE}  
    Default weight.
  \item \texttt{FW\_THIN}  
    Thin weight.
  \item \texttt{FW\_EXTRALIGHT}  
    Extra-light weight.
  \item \texttt{FW\_ULTRALIGHT}  
    Same as \texttt{FW\_EXTRALIGHT}.
  \item \texttt{FW\_LIGHT}  
    Light weight.
  \item \texttt{FW\_NORMAL}  
    Normal weight.
  \item \texttt{FW\_REGULAR}  
    Same as \texttt{FW\_NORMAL}.
  \item \texttt{FW\_MEDIUM}  
    Medium weight.
  \item \texttt{FW\_SEMIBOLD}  
    Semi-bold weight.
  \item \texttt{FW\_DEMIBOLD}  
    Semi-bold weight.
\end{itemize}
Windows API Guide: TEXTMETRIC Structure

Same As FW_SEMIBOLD.

FW_BOLD
  Bold weight.

FW_EXTRABOLD
  Extra-bold weight.

FW_ULTRABOLD
  Same as FW_EXTRABOLD.

FW_HEAVY
  Heavy weight.

FW_BLACK
  Same as FW_HEAVY.

tmOverhang
  The extra width per string added to the font when synthesizing attributes such as boldface and italics. For a boldface effect, this is the distance by which the overstrike is offset. For an italics effect, this is the amount that the top of the character is sheared past the bottom of the character.

tmDigitizedAspectX
  The horizontal aspect of the device for which the font was designed.

tmDigitizedAspectY
  The vertical aspect of the device for which the font was designed.

tmFirstChar
  The value of the first character defined in the font.

tmLastChar
  The value of the last character defined in the font.

tmDefaultChar
  The value of the character to substitute for characters not present in the font.

tmBreakChar
  The value of the character to be used for work breaks for text justification.

tmItalic
  If zero, the font is not an italic font. If a non-zero value, the font is an italic font.

tmUnderlined
  If zero, the font is not an underlined font. If a non-zero value, the font is an underlined font.

tmStruckOut
  If zero, the font is not a strikeout font. If a non-zero value, the font is a strikeout font.

tmPitchAndFamily
  A bitwise OR combination of exactly one *_PITCH flag specifying the pitch of the font and exactly one FF_* flag specifying the font face family of the font:
  DEFAULT_PITCH
    The default pitch.
  FIXED_PITCH
    Fixed pitch.
  VARIABLE_PITCH
    Variable pitch.
  FF_DECORATIVE
Showy, decorative font face.

**FF_DONTCARE**
Do not care about the font face.

**FF_MODERN**
Modern font face (monospaced, sans serif font).

**FF_ROMAN**
Roman font face (proportional-width, serif font).

**FF_SCRIPT**
Script font face which imitates script handwriting.

**FF_SWISS**
Swiss font face (proportional-width, sans serif font).

### tmCharSet

One of the following flags identifying the character set of the font:

- **ANSI_CHARSET**
  ANSI character set.

- **ARABIC_CHARSET**

- **BALTIC_CHARSET**
  Windows 95, 98: Baltic character set.

- **CHINESEBIG5_CHARSET**
  Chinese Big 5 character set.

- **DEFAULT_CHARSET**
  Default character set.

- **EASTEUROPE_CHARSET**
  Windows 95, 98: Eastern European character set.

- **GB2312_CHARSET**
  GB2312 character set.

- **GREEK_CHARSET**
  Windows 95, 98: Greek character set.

- **HANGEUL_CHARSET**
  HANDEUL character set.

- **HEBREW_CHARSET**

- **JOHAB_CHARSET**
  Windows 95, 98: Johab character set.

- **MAC_CHARSET**
  Windows 95, 98: Mac character set.

- **OEM_CHARSET**
  Original equipment manufacturer (OEM) character set.

- **RUSSIAN_CHARSET**
  Windows 95, 98: Russian character set.

- **SHIFTJIS_CHARSET**
  ShiftJis character set.
SYMBOL_CHARSET
    Symbol character set.

THAI_CHARSET
    Windows NT, 2000: Thai character set.

TURKISH_CHARSET
    Windows 95, 98: Turkish character set.

Constant Definitions

Const FW_DONTCARE = 0
Const FW_THIN = 100
Const FW_EXTRALIGHT = 200
Const FW_ULTRALIGHT = 200
Const FW_LIGHT = 300
Const FW_NORMAL = 400
Const FW_REGULAR = 400
Const FW_MEDIUM = 500
Const FW_SEMIBOLD = 600
Const FW_DEMIBOLD = 600
Const FW_BOLD = 700
Const FW_EXTRABOLD = 800
Const FW_ULTRABOLD = 800
Const FW_HEAVY = 900
Const FW_BLACK = 900
Const DEFAULT_PITCH = 0
Const FIXED_PITCH = 1
Const VARIABLE_PITCH = 2
Const FF_DECORATIVE = 80
Const FF_DONTCARE = 0
Const FF_ROMAN = 16
Const FF_SCRIPT = 64
Const FF_SWISS = 32
Const ANSI_CHARSET = 0
Const ARABIC_CHARSET = 178
Const BALTIC_CHARSET = 186
Const CHINESEBIG5_CHARSET = 136
Const DEFAULT_CHARSET = 1
Const EASTEUROPE_CHARSET = 238
Const GB2312_CHARSET = 134
Const GREEK_CHARSET = 161
Const HANGEUL_CHARSET = 129
Const HEBREW_CHARSET = 177
Const JOHAB_CHARSET = 130
Const MAC_CHARSET = 77
Const OEM_CHARSET = 255
Const RUSSIAN_CHARSET = 204
Const SHIFTJIS_CHARSET = 128
Const SYMBOL_CHARSET = 2
Const THAI_CHARSET = 222
Const TURKISH_CHARSET = 162

**Used By**

EnumFontFamExProc, EnumFontFamProc
**TIME_ZONE_INFORMATION Structure**

Type `TIME_ZONE_INFORMATION`
- `Bias` As `Long`
- `StandardName(0 To 31)` As `Integer`
- `StandardDate` As `SYSTEMTIME`
- `StandardBias` As `Long`
- `DaylightName(0 To 31)` As `Integer`
- `DaylightDate` As `SYSTEMTIME`
- `DaylightBias` As `Long`

End Type

`TIME_ZONE_INFORMATION`-type variables hold information about the system's selected time zone. The two arrays in the structure are actually strings, each element holding the ASCII codes for each character (the end of the string is marked by a NULL character, ASCII code 0). For more information about how to convert the arrays into usable data, see the example for `GetTimeZoneInformation`.

**Bias**
The difference in minutes between UTC (a.k.a. GMT) time and local time. It satisfies the formula UTC time = local time + `Bias`.

**StandardName(0 To 31)**
Holds the name of the time zone for standard time.

**StandardDate**
The relative date for when daylight savings time ends.

**StandardBias**
A number to add to `Bias` to form the true bias during standard time.

**DaylightName(0 To 31)**
Holds the name of the time zone for daylight savings time.

**DaylightDate**
The relative date for when daylight savings time begins.

**DaylightBias**
A number to add to `Bias` to form the true bias during daylight savings time.

**Used by:** `GetTimeZoneInformaton`
TOGGLEKEYS Structure

Type TOGGLEKEYS
    cbSize As Long
    dwFlags As Long
End Type

TOGGLEKEYS-type variables store information about the ToggleKeys accessibility feature. ToggleKeys plays a sound whenever the user toggles Caps Lock, Num Lock, or Scroll Lock. This structure holds the settings and properties of ToggleKeys.

cbSize
    The size in bytes of the structure.

dwFlags
    Zero or more of the following flags specifying various settings and properties of ToggleKeys:
    TKF_AVAILABLE = &H2
        The ToggleKeys accessibility feature is available.
    TKF_CONFIRMHOTKEY = &H8
        Win 95/98 only: Open a confirmation dialog box when the user enables ToggleKeys via the hot key,
    TKF_HOTKEYACTIVE = &H4
        Enable the user to toggle ToggleKeys via the hot key: hold Num Lock for eight seconds.
    TKF_HOTKEYSOUND = &H10
        Play a sound when the user toggles ToggleKeys via the hot key.
    TKF_INDICATOR = &H20
        Win 95/98 only: Display an icon in the system tray while ToggleKeys is on.
    TKF_TOGGLEKEYSON = &H1
        ToggleKeys is currently on.

Used by: SystemParametersInfo

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information. Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/t/togglekeys.html
TPMPARAMS Structure

Type TPMPARAMS
    cbSize As Long
    rcExclude As RECT
End Type

Description & Usage

The TPMPARAMS structure contains information needed to properly position a popup menu. Namely, this information consists of an exclusion rectangle. The exclusion rectangle is a rectangle on the screen which the popup menu is not allowed to cover. The popup menu's position will be adjusted in order to avoid the exclusion rectangle.

Visual Basic-Specific Issues

None.

Data Members

cbSize
    The size in bytes of the structure.
rcExclude
    The coordinates of the exclusion rectangle.

Used By

TrackPopupMenuEx

Back to the Structure list.
Back to the Reference section.
Windows API Guide: TPMPARAMS Structure

Last Modified: June 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/t/tpmparams.html
ULARGE_INTEGER Structure

Type ULARGE_INTEGER
   LowPart As Long
   HighPart As Long
End Type

Description & Usage

The ULARGE_INTEGER structure stores a 64-bit unsigned integer. Unsigned integers range in value from &H0 to &HFFFFFFFFFFFFFFF (or in decimal, 2^64). This structure is defined primarily for languages (such as Visual Basic but including many others) which have no intrinsic support for 64-bit unsigned integers. The structure splits the value into two 32-bit halves: a high-order half and a low-order half. If the programming language you use does support 64-bit unsigned integers, you can replace the two data members of this structure with a single value (or use that data type instead of the structure altogether).

Visual Basic-Specific Issues

There is a way to use a variable of the Currency data type to represent a 64-bit integer. For more information, read the article "Faking 64-bit Integers."

Data Members

LowPart
   The 32-bit low-order half of the full 64-bit unsigned integer.

HighPart
   The 32-bit high-order half of the full 64-bit unsigned integer.

Used By

GetDiskFreeSpaceEx, MEMORYSTATUSEX, SHQUERYRBINFO
VS_FIXEDFILEINFO Structure

Type VS_FIXEDFILEINFO
    dwSignature As Long
    dwStrucVersion As Long
    dwFileVersionMS As Long
    dwFileVersionLS As Long
    dwProductVersionMS As Long
    dwProductVersionLS As Long
    dwFileFlagsMask As Long
    dwFileFlags As Long
    dwFileOS As Long
    dwFileType As Long
    dwFileSubtype As Long
    dwFileVersionMS As Long
    dwFileDateMS As Long
End Type

Description & Usage

The VS_FIXEDFILEINFO structure holds version information that describes a 32-bit executable-type file. No version information strings are stored in the structure.

Visual Basic-Specific Issues

None.

Data Members

    dwSignature
        The value &HFEEF04BD.

    dwStrucVersion
        The version number of this structure. The high-order word holds the major version number, and
the low-order word holds the minor version number.

\textit{dwFileVersionMS}

The high-order \textit{dword} of the 64-bit version number. The high-order word of this dword is the leftmost component of the version number, and the low-order word is the number immediately to the right of it. (This is pretty much the major and minor version numbers of the file.)

\textit{dwFileVersionLS}

The low-order dword of the 64-bit version number. The low-order word of this dword is the rightmost component of the version number, and the high-order dword is the component immediately to the left of it. (This is pretty much a major and minor revision number, if it is used at all.)

\textit{dwProductVersionMS}

The high-order dword of the 64-bit version number of the product that the file was shipped with. The format of this value is the same as that of \textit{dwFileVersionMS}.

\textit{dwProductVersionLS}

The low-order dword of the 64-bit version number of the product that the file was shipped with. The format of this value is the same as that of \textit{dwFileVersionLS}.

\textit{dwFileFlagsMask}

A bitmask that specifies what bits of \textit{dwFileFlags} are valid.

\textit{dwFileFlags}

A combination of the following flags specifying additional information about the file's version. Perform a bitwise And between this value and \textit{dwFileFlagsMask} before checking for any of the following flags.

\textbf{VS_FF_DEBUG}

The file contains debugging information.

\textbf{VS_FF_INFOINFERRED}

The version information in this structure was not found inside the file, but instead was created when needed based on the best information available. Therefore, this structure's information may differ slightly from what the "real" values are.

\textbf{VS_FF_PATCHED}

The file has been modified somehow and is not identical to the original file that shipped with the product.

\textbf{VS_FF_PRERELEASE}

The file is a prerelease development version, not a final commercial release.

\textbf{VS_FF_PRIVATEBUILD}

The file was not built using standard release procedures. There should be data in the file's "PrivateBuild" version information string.

\textbf{VS_FF_SPECIALBUILD}

The file was built using standard release procedures, but is somehow different from the normal file having the same version number. There should be data in the file's "SpecialBuild" version information string.

\textit{dwFileOS}

One of the following flags specifying the operating system which the file was meant to run on:

\textbf{VOS_DOS}
The file was designed to run under MS-DOS.

VOS_NT
The file was designed to run under Windows NT/2000.

VOS__WINDOWS16
The file was designed to run under the 16-bit Windows API.

VOS__WINDOWS32
The file was designed to run under the 32-bit Windows API.

VOS_OS216
The file was designed to run under a 16-bit version of OS/2.

VOS_OS232
The file was designed to run under a 32-bit version of OS/2.

VOS__PM16
The file was designed to be run under a 16-bit version of Presentation Manager.

VOS__PM32
The file was designed to be run under a 32-bit version of Presentation Manager.

VOS_UNKNOWN
The operating system under which the file was designed to run could not be determined.

dwFileType
One of the following flags specifying the type of file this is:

VFT_APP
The file is an application.

VFT_DLL
The file is a Dynamic Link Library (DLL).

VFT_DRV
The file is a device driver. dwFileSubtype contains more information.

VFT_FONT
The file is a font. dwFileSubtype contains more information.

VFT_STATIC_LIB
The file is a static link library.

VFT_VXD
The file is a virtual device.

VFT_UNKNOWN
The type of file could not be determined.

dwFileSubtype
Specifies additional information about the file. The usage of this element of the structure depends on the value of dwFileType.

If dwFileType is VFT_DRV, then this is one of the following flags specifying the type of driver:

VFT2_DRV_COMM
The file is a communications driver.

VFT2_DRV_DISPLAY
The file is a display driver.

VFT2_DRV_INSTALLABLE
The file is an installable driver.

VFT2_DRV_KEYBOARD
The file is a keyboard driver.

VFT2_DRV_LANGUAGE
The file is a language driver.

VFT2_DRV_MOUSE
The file is a mouse driver.

VFT2_DRV_NETWORK
The file is a network driver.

VFT2_DRV_PRINTER
The file is a printer driver.

VFT2_DRV_SOUND
The file is a sound driver.

VFT2_DRV_SYSTEM
The file is a system driver.

VFT2_UNKNOWN
The type of driver could not be determined.

If `dwFileType` is VFT_FONT, then this is one of the following flags specifying the type of font:

VFT2_FONT_RASTER
The file is a raster font.

VFT2_FONT_TRUETYPE
The file is a TrueType font.

VFT2_FONT_VECTOR
The file is a vector font.

VFT2_UNKNOWN
The type of font could not be determined.

`dwFileDateMS`  
The high-order dword of the file's 64-bit creation date and time stamp.

`dwFileDateLS`  
The low-order dword of the file's 64-bit creation date and time stamp.

**Constant Definitions**

```
Const VS_FF_DEBUG = &H1
Const VS_FF_INFOINFERRED = &H10
Const VS_FF_PATCHED = &H4
Const VS_FF_PRERELEASE = &H2
Const VS_FF_PRIVATEBUILD = &H8
Const VOS_DOS = &H10000
Const VOS_NT = &H40000
```
Const VOS__WINDOWS16 = &H1
Const VOS__WINDOWS32 = &H4
Const VOS_OS216 = &H20000
Const VOS_OS232 = &H30000
Const VOS__PM16 = &H2
Const VOS__PM32 = &H3
Const VOS_UNKNOWN = &H0
Const VFT_APP = &H1
Const VFT_DLL = &H2
Const VFT_DRV = &H3
Const VFT_FONT = &H4
Const VFT_STATIC_LIB = &H7
Const VFT_VXD = &H5
Const VFT_UNKNOWN = &H0
Const VFT2_DRV_COMM = &HA
Const VFT2_DRV_DISPLAY = &H4
Const VFT2_DRV_INSTALLABLE = &H8
Const VFT2_DRV_KEYBOARD = &H2
Const VFT2_DRV_LANGUAGE = &H3
Const VFT2_DRV_MOUSE = &H5
Const VFT2_DRV_NETWORK = &H6
Const VFT2_DRV_PRINTER = &H1
Const VFT2_DRV_SOUND = &H9
Const VFT2_DRV_SYSTEM = &H7
Const VFT2_FONT_RASTER = &H1
Const VFT2_FONT_TRUETYPE = &H3
Const VFT2_FONT_VECTOR = &H2
Const VFT2_UNKNOWN = &H0

Used By

VerQueryValue

Back to the Structure list.
Back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
WAVEOUTCAPS Structure

Type WAVEOUTCAPS
    wMid As Integer
    wPid As Integer
    vDriverVersion As Long
    szPname As String * 32
    dwFormats As Long
    wChannels As Integer
    dwSupport As Long
End Type

WAVEOUTCAPS-type variables store information about a waveform output device's capabilities as well as other information about it. The various members of the structure identify the abilities of the device.

wMid
    The manufacturer identifier of the maker of the device.

wPid
    The product identifier of the device.

vDriverVersion
    The major and minor version numbers of the device. The high-order half of the value contains the major version number; the low-order half contains the minor version number.

szPname
    A null-terminated string specifying the name of the device.

dwFormats
    Zero or more of the following flags specifying the various audio output formats the device supports:
    WAVE_FORMAT_1M08 = &H1
        Supports 11.025 kHz, 8-bit, mono playback.
    WAVE_FORMAT_1M16 = &H4
        Supports 11.025 kHz, 16-bit, mono playback.
    WAVE_FORMAT_1S08 = &H2
        Supports 11.025 kHz, 8-bit, stereo playback.
    WAVE_FORMAT_1S16 = &H8
        Supports 11.025 kHz, 16-bit, stereo playback.
WAVE_FORMAT_2M08 = &H10
Supports 22.05 kHz, 8-bit, mono playback.

WAVE_FORMAT_2M16 = &H40
Supports 22.05 kHz, 16-bit, mono playback.

WAVE_FORMAT_2S08 = &H20
Supports 22.05 kHz, 8-bit, stereo playback.

WAVE_FORMAT_2S16 = &H80
Supports 22.05 kHz, 16-bit, stereo playback.

WAVE_FORMAT_4M08 = &H100
Supports 44.1 kHz, 8-bit, mono playback.

WAVE_FORMAT_4M16 = &H400
Supports 44.1 kHz, 16-bit, mono playback.

WAVE_FORMAT_4S08 = &H200
Supports 44.1 kHz, 8-bit, stereo playback.

WAVE_FORMAT_4S16 = &H800
Supports 44.1 kHz, 16-bit, stereo playback.

*WChannels*
The number of audio channels on the device. 1 means a mono device; 2 means a stereo device.

*dwSupport*
Zero or more of the following flags specifying which features the device supports:

WAVECAPS_LRVOLUME = &H8
Supports separate left and right channel volumes.

WAVECAPS_PITCH = &H1
Supports pitch control.

WAVECAPS_PLAYBACKRATE = &H2
Supports playback rate control.

WAVECAPS_SAMPLEACCURATE = &H20
Supports returning of sample-accurate position information.

WAVECAPS_SYNC = &H10
Supports synchronous playback -- i.e., it will block while playing buffered audio.

WAVECAPS_VOLUME = &H4
Supports volume control.

**Used by:** [waveOutGetDevCaps](http://216.26.168.92/vbapi/ref/w/waveoutcaps.html)

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/waveoutcaps.html
WIN32_FIND_DATA

Type WIN32_FIND_DATA
   dwFileAttributes As Long
   ftCreationTime As FILETIME
   ftLastAccessTime As FILETIME
   ftLastWriteTime As FILETIME
   nFileSizeHigh As Long
   nFileSizeLow As Long
   dwReserved0 As Long
   dwReserved1 As Long
   cFileName As String * 260
   cAlternate As String * 14
End Type

WIN32_FIND_DATA-type variables hold information found about a file from a file search operation. This information includes the file's attributes; its creation, last-access, and last-modified times; the size of the file; its long filename; and its short filename. The file size is a 64-bit value split into two halves of 32 bits each: a high-order and a low-order half. The actual file size can be found by concatenating the binary or hexadecimal equivalents of the two halves. It can also be found by using the formula:

\[
\text{filesize} = n\text{FileSizeHigh} \times 2^{32} + n\text{FileSizeLow}.
\]

dwFileAttributes

One or more of the following flags identifying the file's attributes:

- **FILE_ATTRIBUTE_ARCHIVE** = &H20
  An archive file (which most files are).
- **FILE_ATTRIBUTE_COMPRESSED** = &H800
  A file residing in a compressed drive or directory.
- **FILE_ATTRIBUTE_DIRECTORY** = &H10
  A directory instead of a file.
- **FILE_ATTRIBUTE_HIDDEN** = &H2
  A hidden file, not normally visible to the user.
- **FILE_ATTRIBUTE_NORMAL** = &H80
  An attribute-less file (cannot be combined with other attributes).
- **FILE_ATTRIBUTE_READONLY** = &H1
  A read-only file.
FILE_ATTRIBUTE_SYSTEM = &H4
   A system file, used exclusively by the operating system.

*ftCreationTime
   The time and date of when the file was created.

*ftLastAccessTime
   The time and date of when the file was last accessed.

*ftLastWriteTime
   The time and date of when the file was last modified or written to.

*nFileSizeHigh
   The high-order half of the file size.

*nFileSizeLow
   The low-order half of the file size.

*dwReserved0
   Reserved for future use.

*dwReserved1
   Reserved for future use.

*cFileName
   The long filename of the file.

*cAlternate
   The short filename of the file.

Used by: FindFirstFile, FindNextFile

Go back to the alphabetical Structure listing.
Go back to the Reference section index.
WNDCLASS Structure

Type WNDCLASS
    style As Long
    lpfnWndProc As Long
    cbClsExtra As Long
    cbWndExtra As Long
    hInstance As Long
    hIcon As Long
    hCursor As Long
    hbrBackground As Long
    lpszMenuName As String
    lpszClassName As String
End Type

Description & Usage

The WNDCLASS structure holds most of the information defining a window class. This information is used by any windows which belong to the class. The only item which this structure does not hold is a handle to the class's small icon. (The more advanced WNDCLASSEX structure does.)

Visual Basic-Specific Issues

None.

Data Members

style

A combination of the following flags specifying various styles to give to windows which belong to the class:

CS_BYTEALIGNCLIENT
    Align the window's client area horizontally on the byte boundary.

CS_BYTEALIGNWINDOW
Align the window horizontally on the byte boundary.

**CS_CLASSDC**
Have all windows belonging to the class use a single shared device context used by all threads.

**CS_DBLCLKS**
Send a double-click message to the window procedure whenever the user double-clicks the mouse inside the window.

**CS_GLOBALCLASS**
Allow any application to create a window from this class. (If this flag is not specified, only the instance registering the class is allowed to use it.) The window class must be defined in a dynamic link library loaded at startup.

**CS_HREDRAW**
Redraw the entire window if a movement or size change alters the width of the window's client area.

**CS_NOCLOSE**
Disable the Close command on the window's menu.

**CS_OWNDC**
Give each window belonging to the class its own device context.

**CS_PARENTDC**
Set the clipping region of a child window equal to that of its parent window, allowing the child to draw on the parent.

**CS_SAVEBITS**
Save as a bitmap the portion of the screen image obscured by the window. This bitmap is then used to redraw the screen when the window is removed. Using this consumes more resources.

**CS_VREDRAW**
Redraw the entire window if a movement or size change alters the height of the window's client area.

**lpfnWndProc**
A pointer to the WindowProc hook function used for the window procedure which processes all of messages of each window belonging to the class.

**cbClsExtra**
The number of extra bytes to allocate following the window class structure for additional storage.

**cbWndExtra**
The number of extra bytes to allocate following the window instance structure for additional storage.

**hInstance**
A handle to the instance which contains the window procedure for the class.

**hIcon**
A handle to the window class's icon.

**hCursor**
A handle to the window class's cursor.

**hbrBackground**
A handle to the brush used to fill the window's background area. This can also be one of the following flags specifying the system color to use as the background color:

- **COLOR_ACTIVEBORDER**
  The color of an active window's border.

- **COLOR_ACTIVECAPTION**
  The color of an active window's caption.

- **COLOR_APPWORKSPACE**
  The color of an application's workspace.

- **COLOR_BACKGROUND**
  The color of the desktop background.

- **COLOR_BTNFACE**
  The color of a button's face.

- **COLOR_BTNHIGHLIGHT**
  The color of a button's highlight area.

- **COLOR_BTNSHADOW**
  The color of a button's shadow area.

- **COLOR_BTNTEXT**
  The color of a button's text.

- **COLOR_CAPTIONTEXT**
  The color of an active window's caption area's text.

- **COLOR_GRAYTEXT**
  The color of grayed-out text.

- **COLOR_HIGHLIGHT**
  The color of a highlight.

- **COLOR_HIGHLIGHTTEXT**
  The color of text in a highlight.

- **COLOR_INACTIVEBORDER**
  The color of an inactive window's border.

- **COLOR_INACTIVECAPTION**
  The color of an inactive window's caption.

- **COLOR_INACTIVECAPTIONTEXT**
  The color of an inactive window's caption area's text.

- **COLOR_MENU**
  The color of a menu.

- **COLOR_MENUTEXT**
  The color of a menu's text.

- **COLOR_SCROLLBAR**
  The color of a scrollbar.

- **COLOR_WINDOW**
  The color of a window.

- **COLOR_WINDOWFRAME**
  The color of a window's frame.

- **COLOR_WINDOWTEXT**
The color of a window's text.

`lpszMenuName`
- The name of the window class's menu resource, if any.

`lpszClassName`
- The name of the window class.

**Constant Definitions**

```c
Const CS_BYTEALIGNCLIENT = &H1000
Const CS_BYTEALIGNWINDOW = &H2000
Const CS_CLASSDC = &H40
Const CS_DBLCLKS = &H8
Const CS_HREDRAW = &H2
Const CS_NOCLOSE = &H200
Const CS_OWNDC = &H20
Const CS_PARENTDC = &H80
Const CS_SAVEBITS = &H800
Const CS_VREDRAW = &H1
Const COLOR_ACTIVEBORDER = 10
Const COLOR_ACTIVECAPTION = 2
Const COLOR_APPWORKSPACE = 12
Const COLOR_BACKGROUND = 1
Const COLOR_BTNFACjang = 15
Const COLOR_BTNHIGHLIGHT = 20
Const COLOR_BTNSHADOW = 16
Const COLOR_BTNTEXT = 18
Const COLOR.CaptionText = 9
Const COLOR_GRAYTEXT = 17
Const COLOR.HIGHLIGHT = 13
Const COLOR.HIGHLIGHTTEXT = 14
Const COLOR_INACTIVEBORDER = 11
Const COLOR_INACTIVECAPTION = 3
Const COLOR_INACTIVECAPTIONTEXT = 19
Const COLOR_MENU = 4
Const COLOR_MENUTEXT = 7
Const COLOR_SCROLLBAR = 0
Const COLOR_WINDOW = 5
Const COLOR_WINDOWFRAME = 6
Const COLOR_WINDOWTEXT = 8
```

**Used By**
GetClassInfo, RegisterClass

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

Last Modified: August 22, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wndclass.html
WNDCLASSEX Structure

Type WNDCLASSEX

    cbSize As Long
    style As Long
    lpfnWndProc As Long
    cbClsExtra As Long
    cbWndExtra As Long
    hInstance As Long
    hIcon As Long
    hCursor As Long
    hbrBackground As Long
    lpszMenuName As String
    lpszClassName As String
    hIconSm As Long

End Type

Description & Usage

The WNDCLASSEX structure holds all of the information defining a window class. This information is used by any windows which belong to the class.

Visual Basic-Specific Issues

None.

Data Members

\textit{cbSize} \\
\ 
The size in bytes of the structure.

\textit{style} \\
\ 
A combination of the following flags specifying various styles to give to windows which belong to the class:
CS_BYTEALIGNCLIENT
  Align the window's client area horizontally on the byte boundary.

CS_BYTEALIGNWINDOW
  Align the window horizontally on the byte boundary.

CS_CLASSDC
  Have all windows belonging to the class use a single shared device context used by all threads.

CS_DBLCLKS
  Send a double-click message to the window procedure whenever the user double-clicks the mouse inside the window.

CS_GLOBALCLASS
  Allow any application to create a window from this class. (If this flag is not specified, only the instance registering the class is allowed to use it.) The window class must be defined in a dynamic link library loaded at startup.

CS_HREDRAW
  Redraw the entire window if a movement or size change alters the width of the window's client area.

CS_NOCLOSE
  Disable the Close command on the window's menu.

CS_OWNDC
  Give each window belonging to the class its own device context.

CS_PARENTDC
  Set the clipping region of a child window equal to that of its parent window, allowing the child to draw on the parent.

CS_SAVEBITS
  Save as a bitmap the portion of the screen image obscured by the window. This bitmap is then used to redraw the screen when the window is removed. Using this consumes more resources.

CS_VREDRAW
  Redraw the entire window if a movement or size change alters the height of the window's client area.

lpfnWndProc
  A pointer to the WindowProc hook function used for the window procedure which processes all of messages of each window belonging to the class.

cbClsExtra
  The number of extra bytes to allocate following the window class structure for additional storage.

cbWndExtra
  The number of extra bytes to allocate following the window instance structure for additional storage.

hInstance
  A handle to the instance which contains the window procedure for the class.

hIcon
A handle to the window class's icon.

*hCursor*

A handle to the window class's cursor.

*hbrBackground*

A handle to the brush used to fill the window's background area. This can also be one of the following flags specifying the system color to use as the background color:

COLOR_ACTIVEBORDER
   The color of an active window's border.
COLOR_ACTIVECAPTION
   The color of an active window's caption.
COLOR_APPWORKSPACE
   The color of an application's workspace.
COLOR_BACKGROUND
   The color of the desktop background.
COLOR_BTNFACE
   The color of a button's face.
COLOR_BTNHIGHLIGHT
   The color of a button's highlight area.
COLOR_BTNSHADOW
   The color of a button's shadow area.
COLOR_BTNTEXT
   The color of a button's text.
COLOR_CAPTIONTEXT
   The color of an active window's caption area's text.
COLOR_GRAYTEXT
   The color of grayed-out text.
COLOR_HIGHLIGHT
   The color of a highlight.
COLOR_HIGHLIGHTTEXT
   The color of text in a highlight.
COLOR_INACTIVEBORDER
   The color of an inactive window's border.
COLOR_INACTIVECAPTION
   The color of an inactive window's caption.
COLOR_INACTIVECAPTIONTEXT
   The color of an inactive window's caption area's text.
COLOR_MENU
   The color of a menu.
COLOR_MENUTEXT
   The color of a menu's text.
COLOR_SCROLLBAR
   The color of a scrollbar.
COLOR_WINDOW
The color of a window.

COLOR_WINDOWFRAME
The color of a window's frame.

COLOR_WINDOWTEXT
The color of a window's text.

lpszMenuName
The name of the window class's menu resource, if any.

lpszClassName
The name of the window class.

hIconSm
A handle to the window class's small icon.

Constant Definitions

Const CS_BYTEALIGNCLIENT = &H1000
Const CS_BYTEALIGNWINDOW = &H2000
Const CS_CLASSDC = &H40
Const CS_DBLCLKS = &H8
Const CS_HREDRAW = &H2
Const CS_NOCLOSE = &H200
Const CS_OWNDC = &H20
Const CS_PARENTDC = &H80
Const CS_SAVEBITS = &H800
Const CS_VREDRAW = &H1
Const COLOR_ACTIVEBORDER = 10
Const COLOR_ACTIVECAPTION = 2
Const COLOR_APPWORKSPACE = 12
Const COLOR_BACKGROUND = 1
Const COLOR_BTNFACE = 15
Const COLOR_BTNHIGHLIGHT = 20
Const COLOR_BTNSHADOW = 16
Const COLOR_BTNTEXT = 18
Const COLOR_CAPTIONTEXT = 9
Const COLOR_GRAYTEXT = 17
Const COLOR_HIGHLIGHT = 13
Const COLOR_HIGHLIGHTTEXT = 14
Const COLOR_INACTIVEBORDER = 11
Const COLOR_INACTIVECAPTION = 3
Const COLOR_INACTIVECAPTIONTEXT = 19
Const COLOR_MENU = 4
Const COLOR_MENUTEXT = 7
Const COLOR_SCROLLBAR = 0
Const COLOR_WINDOW = 5
Const COLOR_WINDOWFRAME = 6
Const COLOR_WINDOWTEXT = 8

**Used By**

GetClassInfoEx, RegisterClassEx

Go back to the alphabetical Structure listing.
Go back to the Reference section index.

---

**Last Modified:** August 22, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com [Send Encrypted E-Mail](#)
This page is at [http://www.vbapi.com/ref/w/wndclassex.html](http://www.vbapi.com/ref/w/wndclassex.html)
WSADATA Structure

Type WSADATA

    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long

End Type

Description & Usage

The WSADATA structure stores information about the Windows Sockets (Winsock) implementation being used by your program. Most of the structure's contents refer to the version of Winsock the program is interfacing with.

Visual Basic-Specific Issues

None.

Data Members

wVersion

    The version of Winsock that Windows expects you to use. The low-order byte contains the major version number, and the high-order byte contains the minor version number. This should be the same version you specified when calling WSAStartup.

wHighVersion

    The highest version of Winsock that Windows supports. The version number is encoded in the same manner as wVersion.

szDescription

    A null-terminated string that describes the Winsock implementation being used.
**szSystemStatus**
A null-terminated string specifying the Winsock implementation's current status.

**iMaxSockets**
Not used -- this member exists solely for backwards compatibility with Winsock version 1.1 and earlier.

**iMaxUdpDg**
Not used -- this member exists solely for backwards compatibility with Winsock version 1.1 and earlier.

**lpVenderInfo**
Not used -- this member exists solely for backwards compatibility with Winsock version 1.1 and earlier.

**Used By**

**WSAStartup**

Back to the Structure list.
Back to the Reference section.
Windows API Reference: Messages

Last Update: January 21, 2001
Number of Messages Listed: 66 (6 added)

Below is a categorical list of the API messages currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to:

- Buttons
- Combo Boxes
- Edit Controls
- IP Address Control
- List Boxes
- Media Control Interface (MCI)
- Menus
- Mouse
- Timers
- Windows

- Buttons
  - BM_CLICK
  - BM_GETCHECK
  - BM_GETSTATE
  - BM_SETCHECK
  - BM_SETSTATE

- Combo Boxes
  - CB_ADDSTRING
CB_DELETESTRING
CB_GETCOUNT
CB_GETCURSEL
CB_GETDROPPEDSTATE
CB_GETLBTEXT
CB_GETLBTEXTLEN
CB_INSERTSTRING
CB_RESETCONTENT
CB_SETCURSEL
CB_SHOWDROPDOWN

Edit Controls
EM_CANUNDO
EM_GETFIRSTVISIBLELINE
EM_GETLINE
EM_GETPASSWORDCHAR
EM_GETSEL
EM_LINEINDEX
EM_LINELENGTH
EM_REPLACESEL
EM_SETPASSWORDCHAR
EM_SETSEL
EM_UNDO

IP Address Control
IPM_CLEARADDRESS
IPM_GETADDRESS
IPM_ISBLANK
IPM_SETADDRESS
IPM_SETFOCUS
IPM_SETRANGE

List Boxes
LB_ADDSTRING
LB_DELETESTRING
LB_GETCOUNT
LB_GETCURSEL NEW
LB_GETSEL NEW
LB_GETSELCOUNT NEW
LB_GETSELCITEMS NEW
LB_GETTEXT
LB_GETTEXTLEN
- **LB_INSERTSTRING**
- **LB_RESETCONTENT**
- **LB_SETCURSEL** NEW
- **LB_SETSEL** NEW

- **Media Control Interface (MCI)**
  - **MM_MCINOTIFY**

- **Menus**
  - **WM_COMMAND**
  - **WM_INITMENU**
  - **WM_SYSCOMMAND**

- **Mouse**
  - **WM_LBUTTONDOWN**
  - **WM_LBUTTONUP**
  - **WM_MBUTTONDOWN**
  - **WM_MBUTTONUP**
  - **WM_MOUSEMOVE**
  - **WM_RBUTTONDOWN**
  - **WM_RBUTTONUP**
  - **WM_RBUTTONDBLCLK**
  - **WM_MBUTTONDBLCLK**

- **Timers**
  - **WM_TIMER**

- **Windows**
  - **WM_CLOSE**
  - **WM_GETTEXT**
  - **WM_GETTEXTLENGTH**
  - **WM_HELP**
  - **WM_SETTEXT**

---

Go back to the Reference section index.

---

**Last Modified:** January 21, 2001

This page is copyright © 2001 Paul Kuliniewicz. [Copyright Information]  [Revised October 29, 2000]

Go back to the Windows API Guide home page.
BM_CLICK Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.5 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the BM_CLICK message to a button simulates the user clicking on that button. Sometimes this message will not work if the button's parent window is not active. To avoid this, use SetActiveWindow to make its parent window active before sending this message.

Return Value

This message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to 0.

lParam
Not used -- set to 0.

Constant Definitions

Const BM_CLICK = &HF5

Example

This code uses a Timer object to periodically check for a dialog box to appear. If found, the code uses BM_CLICK to "click"
the button labeled "Resume" to close the dialog box. Something akin to this code might make a nifty way to automatically clear out those annoying "Are you still online?" dialog boxes that some ISP connection programs open every 45 minutes or so. To use this code, place a timer control named Timer1 in a window and give it a period of something like 10000 (every 10 seconds).

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const BM_CLICK = &HF5
Public Declare Function SetActiveWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" (ByVal lpClassName As Any, ByVal lpWindowName As Any) As Long
Public Declare Function FindWindowEx Lib "user32.dll" Alias "FindWindowExA" (ByVal hwndParent As Long, ByVal hwndChildAfter As Long, ByVal lpszClass As Any, ByVal lpszWindow As Any) As Long

' *** Place the following code in the form. ***
Private Sub Timer1_Timer()
    Dim hwndDialog As Long  ' handle to the dialog box
    Dim hwndButton As Long  ' handle to the Resume button
    Dim retval As Long      ' return value

    ' First, see if the dialog box (titled "Inactivity Warning" is currently open.
    hwndDialog = FindWindow(CLng(0), "Inactivity Warning")
    If hwndDialog = 0 Then Exit Sub

    ' Now get a handle to the "Resume" button in the dialog.
    hwndButton = FindWindowEx(hwndDialog, 0, CLng(0), "Resume")

    ' After making sure that the dialog box is the active window, click "Resume".
    retval = SetActiveWindow(hwndDialog)
    retval = SendMessage(hwndButton, BM_CLICK, ByVal CLng(0), ByVal CLng(0))
End Sub

Category

Buttons

Back to the Messagelist.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
BM_GETCHECK Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the `BM_GETCHECK` message to a button determines if that button is currently checked. Obviously, this message only works with buttons that are check boxes or radio boxes.

Return Value

If an error occurred, the message returns zero. Otherwise, the message returns one of the following values specifying the button’s checked state:

- **BST_CHECKED**: The button is checked.
- **BST_INDETERMINATE**: The button is grayed, in an indeterminate state. This only works with check boxes that have three possible states.
- **BST_UNCHECKED**: The button is unchecked.

Visual Basic-Specific Issues

None.

Parameters

- `wParam`: Not used -- set to 0.
- `lParam`: Not used -- set to 0.

Constant Definitions
Example

Use the **BM_GETCHECK** message to determine the checked status of check box Check1 when a button is pressed. To run this example, you need to create a check box control named Check1 and a command button named Command1 in a form window. The check box may be any type you wish.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const BM_GETCHECK = &HF0
Public Const BST_CHECKED = &1
Public Const BST_INDETERMINATE = &2
Public Const BST_UNCHECKED = &0

' *** Place the following code inside the form window. ***
Private Sub Command1_Click()
    Dim state As Long  ' checked state of the check box
    ' Find the checked state of Check1 and tell the user what it is.
    state = SendMessage(Check1.hWnd, BM_GETCHECK, ByVal CLng(0), ByVal CLng(0))
    Select Case state
    Case BST_CHECKED
        Debug.Print "The check box is checked."
    Case BST_INDETERMINATE
        Debug.Print "The check box is in its third state (grayed)."
    Case BST_UNCHECKED
        Debug.Print "The check box is not checked."
    End Select
End Sub

See Also

**BM_GETSTATE**, **BM_SETCHECK**

Category

**Buttons**

**Back to the Function list.**
BM_GETSTATE Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the BM_GETSTATE message to a button retrieves information about the state of the button. This message works will all types of buttons, including command buttons, check boxes, and radio buttons.

Return Value

If an error occured, the message returns zero. Otherwise, the message returns a combination of the following values specifying the button’s current state:

- **BST_CHECKED**: The button is checked.
- **BST_FOCUS**: The button has the keyboard focus.
- **BST_INDETERMINATE**: The button is grayed, in an indeterminate state. This only works with check boxes that have three possible states.
- **BST_PUSHED**: The button is being pushed. This is typically caused by the left button being held down on top of the button, making it look pushed in.
- **BST_UNCHECKED**: The button is unchecked.

Visual Basic-Specific Issues

None.

Parameters

*wParam*

Not used -- set to 0.
Constant Definitions

Const BM_GETSTATE = &HF2
Const BST_CHECKED = &H1
Const BST_FOCUS = &H8
Const BST_INDETERMINATE = &H2
Const BST_PUSHED = &H4
Const BST_UNCHECKED = &H0

Example

Use the BM_GETSTATE message to determine the state of check box Check1 when a button is pressed. To run this example, you need to create a check box control named Check1 and a command button named Command1 in a form window. The check box may be any type you wish.

This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const BM_GETSTATE = &HF2
Public Const BST_CHECKED = &H1
Public Const BST_FOCUS = &H8
Public Const BST_INDETERMINATE = &H2
Public Const BST_PUSHED = &H4
Public Const BST_UNCHECKED = &H0

*** Place the following code inside the form window. ***
Private Sub Command1_Click()
Dim state As Long  ' checked state of the check box

' Find the state of Check1 and tell the user what it is.
state = SendMessage(Check1.hWnd, BM_GETSTATE, ByVal CLng(0), ByVal CLng(0))
If state And BST_CHECK Then
    Debug.Print "The check box is checked."
ElseIf state And BST_INDETERMINATE Then
    Debug.Print "The check box is in its third state (grayed)."
Else
    Debug.Print "The check box is not checked."
End If
If state And BST_FOCUS Then Debug.Print "The check box has the keyboard focus."
    If state And BST_PUSHED Then Debug.Print "The check box is being pushed."
End Sub
See Also

BM_GETCHECK, BM_SETSTATE

Category

Buttons

Back to the Function list.
Back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/b/bm_getstate.html
BM_SETCHECK Message

Platforms

- **Windows 95:** Supported.
- **Windows 98:** Supported.
- **Windows NT:** Requires Windows NT 3.1 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

Sending the **BM_SETCHECK** message to a button changes the checked state of a button. Naturally, the button has to be either a check box or a radio button for the message to work. The message can check or uncheck the button. If the button is a three-state check box, the message can also put the button into its third, grayed state.

Return Value

The **BM_SETCHECK** message always returns 0.

Visual Basic-Specific Issues

None.

Parameters

`wParam`

One of the following flags specifying how to set the button's checked state:

- **BST_CHECKED**
  - Check the button.
- **BST_INDETERMINATE**
  - Put the button into its third, grayed state. This only works for three-state check boxes.
- **BST_UNCHECKED**
  - Uncheck the button.

`lParam`

Not used -- set to 0.

Constant Definitions
Const BM_SETCHECK = &HF1
Const BST_CHECKED = &1
Const BST_INDETERMINATE = &2
Const BST_UNCHECKED = &0

Example

Use the BM_SETCHECK message to place a check mark inside of Check1. Do this when a button is pressed. To run this example, place a check box control named Check1 and a command button named Command1 inside a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const BM_SETCHECK = &HF1
Public Const BST_CHECKED = &1
Public Const BST_INDETERMINATE = &2
Public Const BST_UNCHECKED = &0

' *** Place the following code inside the form window. ***
Private Sub Command1_Click()
    Dim retval As Long  ' return value
    ' Place a check mark inside Check1.
    retval = SendMessage (Check1.hWnd, BM_SETCHECK, ByVal CLng(BST_CHECKED), ByVal CLng(0))
    ' That's it!
End Sub

See Also

BM_GETCHECK, BM_SETSTATE

Category

Buttons

Back to the Function list.
Back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
Windows API Guide: BM_SETCHECK Message

E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/b/bm_setcheck.html
BM_SETSTATE Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **BM_SETSTATE** message changes the pushed state of a button. This message allows you to control whether a button looks pushed down or not. The appearance of the button has no other effect on its state or checked/unchecked value, if applicable. In reality, this message only affects command buttons.

Return Value

The **BM_SETSTATE** message always returns 0.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  
  If this is a non-zero value, the button's appearance is changed to make it looked pushed down. If this is zero, the button's appearance is changed to make it look normal.

- **lParam**
  
  Not used -- set to 0.

Example

Use the **BM_SETSTATE** message to make button Command1 look pressed down. Do this when the user clicks on button Command2. Obviously, to use this example, you need to make two command buttons named Command1 and Command2 on a form window.

' This code is licensed according to the terms and conditions listed here.
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const BM_SETSTATE = &HF4

Private Sub Command2_Click()
    Dim retval As Long  ' return value
    ' Make button Command1 look pressed down.
    retval = SendMessage(Check1.hWnd, BM_SETSTATE, ByVal CLng(1), ByVal CLng(0))
    ' That's it!
End Sub

See Also

BM_GETSTATE, BM_SETCHECK

Category

Buttons

Back to the Function list.
Back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/b/bm_setstate.html
CB_ADDSTRING Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the CB_ADDSTRING message to a combo box control adds a string to the control's drop-down list. If the combo box is sorted, the new string is added in its proper position according to the sort. If the combo box is not sorted, the new string is added to the end. To control where the string is added, use the CB_INSERTSTRING message instead.

Return Value

If successful, the message returns the zero-based index of the newly added string's position in the combo box's drop-down box. If there is insufficient space to store the new string, the message returns CB_ERRSPACE. If some other error occurs, the message returns CB_ERR.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**: Not used -- set to 0.
- **lParam**: The string to add to the combo box's drop-down box.

Constant Definitions

```
Const CB_ADDSTRING = &H143
Const CB_ERR = -1
Const CB_ERRSPACE = -2
```
Example

When the user clicks button Command1, empty the drop-down box of combo box Combo1, and then add three strings to it. The order in which the strings appear will depend on whether the combo box is sorted or not.

To use this example, place a combo box named Combo1 and a command button named Command1 on a form window. To verify that the combo box’s list is being emptied, you may wish to add some items to it before running the example.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_ADDSTRING = &H143
Public Const CB_RESETCONTENT = &H14B

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim retval As Long ' return value

    ' Empty the drop-down box of Combo1.
    retval = SendMessage(Combo1.hWnd, CB_RESETCONTENT, ByVal CLng(0), ByVal CLng(0))

    ' Now add three strings to Combo1. Their exact placement will depend on whether Combo1 is sorted or not.
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "First Item Added")
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "Second Item Added")
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "Last Item Added")
End Sub

See Also

CB_DELETESTRING, CB_INSERTSTRING, CB_RESETCONTENT

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.
CB_DELETESTRING Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the CB_DELETESTRING message to a combo box removes one of the items from its drop-down box.

Return Value

If successful, the message returns the number of items remaining in the combo box's drop-down box. If an error occurred, the message returns CB_ERR.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based index of the string to delete from the combo box's drop-down box.

lParam

Not used -- set to zero.

Constant Definitions

Const CB_DELETESTRING = &H144
Const CB_ERR = -1

Example

When the user clicks button Command1, remove the second item from combo box's Combo1 drop-down box. To run this example, place a combo box named Combo1 and a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_DELETESTRING = &H144
Public Const CB_ERR = -1

*** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim result As Long  ' result of string deletion attempt
    ' Remove the second item from Combo1's drop-down list
    ' box, and display the result.
    result = SendMessage(Combo1.hWnd, CB_DELETESTRING, ByVal CLng(1), ByVal CLng(0))
    If result = CB_ERR Then
        Debug.Print "Unable to delete the second string in Combo1's drop-down box."
    Else
        Debug.Print "There are"; result; "strings left in Combo1's drop-down box."
    End If
End Sub

See Also

CB_ADDSTRING, CB_INSERTSTRING, CB_RESETCONTENT

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/cb_deletestring.html
CB_GETCOUNT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the CB_GETCOUNT message determines how many items exist in the list box portion of a combo box control. Keep in mind that the list box items are zero-based; the first item has an index of zero, and the last one has an index of the count minus one.

Return Value

If successful, the message returns the number of items that are in the list box portion of a combo box control. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - Not used -- set to 0.
- lParam
  - Not used -- set to 0.

Constant Definitions

Const CB_GETCOUNT = &H146

Example
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_GETCOUNT = &H146
Public Const CB_GETLBTEXT = &H148
Public Const CB_GETLBTEXTLEN = &H149

Display the text of the second-to-last item in the list box portion of combo box Combo1.
Dim count As Long ' number of items in the combo box
Dim s2l As Long ' index of the second-to-last item
Dim itemtext As String ' text of that item
Dim textlen As Long ' length of the item text

Figure out the index of the second-to-last item by subtracting two from the total item count (remember, the index is zero-based).
count = SendMessage(Combo1.hWnd, CB_GETCOUNT, ByVal CLng(0), ByVal CLng(0))
s2l = count - 2

Make the string long enough to receive that item's text.
textlen = SendMessage(Combo1.hWnd, CB_GETLBTEXTLEN, ByVal s2l, ByVal CLng(0))
itemtext = Space(textlen) & vbNullChar
Get the item text and remove the trailing null.
textlen = SendMessage(Combo1.hWnd, CB_GETLBTEXT, ByVal s2l, ByVal itemtext)
itemtext = Left(itemtext, textlen)
Finally, display the result.
Debug.Print "The second-to-last item is "; itemtext

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kulininewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/cb_getcount.html
 CB_GETCURSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The CB_GETCURSEL message finds out which list-box item is currently selected in a combo box control. This message only returns useful information if the combo box's selection is one of the items available in its list box portion. If the user types different data into the text box portion, the function merely reports that no item is selected.

Return Value

The message returns the zero-based index of the selected item (the first item has an index of 0, etc.). If none of the items in the list box portion are selected, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

wParam
- Not used -- set to 0.

lParam
- Not used -- set to 0.

Constant Definitions

Const CB_GETCURSEL = &H147

Example
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_GETCURSEL = &H147
Public Const CB_GETLBTEXT = &H148
Public Const CB_GETLBTEXTLEN = &H149

' Display the text of whatever item in combo box Combo1 is currently selected. If no list box item is selected, say so.
Dim index As Long ' index to the selected item
Dim itemtext As String ' the text of the selected item
Dim textlen As Long ' the length of the selected item's text

' Determine the index of the selected item.
index = SendMessage(Combo1.hWnd, CB_GETCURSEL, ByVal CLng(0), ByVal CLng(0))
' Decide what to do based on that.
Select Case index
Case -1 ' No list box item was selected.
  Debug.Print "No list box item in the combo box is selected."
Case Else ' Some item is selected.
  ' Determine how long the item's text is.
  textlen = SendMessage(Combo1.hWnd, CB_GETLBTEXTLEN, ByVal CLng(index), ByVal CLng(0))
  ' Make enough room in the string to receive the text, including the terminating null.
  itemtext = Space(textlen) & vbNullChar
  ' Retrieve that item's text and display it.
  textlen = SendMessage(Combo1.hWnd, CB_GETLBTEXT, ByVal CLng(index), ByVal itemtext)
  itemtext = Left(itemtext, textlen)
  Debug.Print "Selected item: "; itemtext
End Select

See Also

CB_SETCURSEL

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.
CB_GETDROPPEDSTATE Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the CB_GETDROPPEDSTATE message to a combo box determines if its list box portion is currently visible. In most combo boxes, the list box can be either dropped down or hidden. This message only works if the combo box has that capability.

Return Value

If the message returns 0, the list box portion is currently hidden. If the message returns a non-zero value, the list box is visible.

Visual Basic-Specific Issues

None.

Parameters

* `wParam`: Not used -- set to 0.
* `lParam`: Not used -- set to 0.

Constant Definitions

```vbnet
Const CB_GETDROPPEDSTATE = &H157
```

Example

```
' This code is licensed according to the terms and conditions listed here.
```
Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_GETDROPPEDSTATE = &H157

' Determine if the list box of combo box control Combo1 is currently
' in its dropped down, visible state.
Dim isdropped As Long  ' is it dropped down or not?
isdropped = SendMessage(Combo1.hWnd, CB_GETDROPPEDSTATE, ByVal CLng(0), ByVal CLng(0))
Select Case isdropped
Case 0
    Debug.Print "The list box portion is not visible."
Case Else
    Debug.Print "The list box portion is visible."
End Select

See Also

CB_SHOWDROPDOWN

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/cb_getdroppedstate.html
CB_GETLBTEXT Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The **CB_GETLBTEXT** message retrieves the text of one of the items in the list box portion of a combo box control.

Return Value

If successful, the message returns the length of the string copied into the string passed as *lParam*, not including the terminating null character. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  
  The zero-based index of the list box item to retrieve the text of. (The first item has an index of 0, etc.)

- **lParam**
  
  The string to copy the list box item text into. The string must have enough room to receive the entire string along with a terminating null character.

Constant Definitions

```vbnet
Const CB_GETLBTEXT = &H148
```

Example

```
' This code is licensed according to the terms and conditions listed here.
```
' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_GETCURSEL = &H147
Public Const CB_GETLBTEXT = &H148
Public Const CB_GETLBTEXTLEN = &H149

' Display the text of whatever item in combo box Combol is currently selected. If no list box item is selected, say so.
Dim index As Long       ' index to the selected item
Dim itemtext As String  ' the text of the selected item
Dim textlen As Long     ' the length of the selected item's text

' Determine the index of the selected item.
index = SendMessage(Combol.hWnd, CB_GETCURSEL, ByVal CLng(0), ByVal CLng(0))
' Decide what to do based on that.
Select Case index
Case -1    ' No list box item was selected.
    Debug.Print "No list box item in the combo box is selected."
Case Else  ' Some item is selected.
    ' Determine how long the item's text is.
    textlen = SendMessage(Combol.hWnd, CB_GETLBTEXTLEN, ByVal CLng(index), ByVal CLng(0))
    ' Make enough room in the string to receive the text, including the terminating null.
    itemtext = Space(textlen) & vbNullChar
    ' Retrieve that item's text and display it.
    textlen = SendMessage(Combol.hWnd, CB_GETLBTEXT, ByVal CLng(index), ByVal itemtext)
    itemtext = Left(itemtext, textlen)
    Debug.Print "Selected item: "; itemtext
End Select

See Also
CB_GETLBTEXTLEN

Category
Combo Boxes

Back to the Message list.
Back to the Reference section.
CB_GETLBTEXTLEN Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The CB_GETLBTEXTLEN message retrieves the length of the text of one of the items in the list box portion of a combo box control. The reported length does not include its terminating null character. When determining how long to size a string to receive the list box item’s text, remember to add one to include the null.

Return Value

If successful, the message returns the length of the list box item's text, not including the terminating null character. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  
  The zero-based index of the list box item to retrieve the length of the text of. (The first item has an index of 0, etc.)

- **lParam**
  
  Not used -- set to 0.

Constant Definitions

- **Const CB_GETLBTEXTLEN = &H149**

Example

This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_GETCURSEL = &H147
Public Const CB_GETLBTEXT = &H148
Public Const CB_GETLBTEXTLEN = &H149

Display the text of whatever item in combo box Combo1 is currently selected. If no list box item is selected, say so.

Dim index As Long ' index to the selected item
Dim itemtext As String ' the text of the selected item
Dim textlen As Long ' the length of the selected item's text

Determine the index of the selected item.

index = SendMessage(Combo1.hWnd, CB_GETCURSEL, ByVal CLng(0), ByVal CLng(0))

' Decide what to do based on that.
Select Case index
Case -1  ' No list box item was selected.
  Debug.Print "No list box item in the combo box is selected."
Case Else  ' Some item is selected.
  ' Determine how long the item's text is.
  textlen = SendMessage(Combo1.hWnd, CB_GETLBTEXTLEN, ByVal CLng(index), ByVal CLng(0))
  ' Make enough room in the string to receive the text, including the terminating null.
  itemtext = Space(textlen) & vbNullChar
  ' Retrieve that item's text and display it.
  itemtext = Left(itemtext, textlen)
  Debug.Print "Selected item: "; itemtext
End Select

See Also

CB_GETLBTEXT

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.
CB_INSERTSTRING Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **CB_INSERTSTRING** message to a combo box inserts a string into its drop-down box. The string is placed at the position specified in the parameters, regardless of whether the combo box is sorted or not.

Return Value

If successful, the message returns the zero-based index of the newly added string's position in the combo box's drop-down box. If there is insufficient space to store the new string, the message returns CB_ERRSTRING. If some other error occurs, the message returns CB_ERR.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**: The zero-based index of the position to insert the string in the combo box's drop-down box. If this is -1, the string is added to the end of the list.
- **lParam**: The string to add to the combo box's drop-down box.

Constant Definitions

```
Const CB_INSERTSTRING = &H14A
Const CB_ERR = -1
Const CB_ERRSPACE = -2
```
Example

When the user clicks button Command1, insert three strings into combo box Combo1’s drop-down box. One string is added to the beginning, one to the third position, and one to the end. To run this example, place a combo box named Combo1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_INSERTSTRING = &H14A

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim retval As Long  ' return value

    ' Add a string to the beginning of the drop-down box.
    retval = SendMessage(Combo1.hWnd, CB_INSERTSTRING, ByVal CLng(0), ByVal "First Item"

    ' Insert a string at the third position in the drop-down box.
    retval = SendMessage(Combo1.hWnd, CB_INSERTSTRING, ByVal CLng(2), ByVal "Third Item"

    ' Add a string to the end of the drop-down box.
    retval = SendMessage(Combo1.hWnd, CB_INSERTSTRING, ByVal CLng(-1), ByVal "Last Item"

End Sub

See Also

CB_ADDSTRING, CB_DELETESTRING, CB_RESETCONTENT

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: September 24, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
CB_RESETCONTENT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the CB_RESETCONTENT message to a combo box removes all the items from its drop-down box and edit control.

Return Value

This message always returns zero.

Visual Basic-Specific Issues

None.

Parameters

wParam
- Not used -- set to 0.

lParam
- Not used -- set to 0.

Constant Definitions

Const CB_RESETCONTENT = &H14B

Example

When the user clicks button Command1, empty the drop-down box of combo box Combo1, and then add three strings to it. The order in which the strings appear will depend on whether the combo box is sorted or not.
To use this example, place a combo box named Combo1 and a command button named Command1 on a form window. To verify that the combo box's list is being emptied, you may wish to add some items to it before running the example.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_ADDSTRING = &H143
Public Const CB_RESETCONTENT = &H14B

' *** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim retval As Long ' return value

    ' Empty the drop-down box of Combo1.
    retval = SendMessage(Combo1.hWnd, CB_RESETCONTENT, ByVal CLng(0), ByVal CLng(0))

    ' Now add three strings to Combo1. Their exact placement will depend on whether Combo1 is sorted or not.
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "First Item Added")
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "Second Item Added")
    retval = SendMessage(Combo1.hWnd, CB_ADDSTRING, ByVal CLng(0), ByVal "Last Item Added")
End Sub

See Also

CB_ADDSTRING, CB_DELETESTRING, CB_INSERTSTRING

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: Month DD, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
CB_SETCURSEL Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The **CB_SETCURSEL** message chooses which item in the list box portion of a combo box control is selected. Doing so erases the user's previous selection, as well as any other text he may have entered in the text box portion of the control.

Return Value

The message returns the zero-based index of the item that is now selected (the first item has an index of 0, etc.). If an error occurred, or if the message specified a list box item that does not exist, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  - The zero-based index of the list box item to select.
- **lParam**
  - Not used -- set to 0.

Constant Definitions

`Const CB_SETCURSEL = &H14E`

Example

```vbnet
' This code is licensed according to the terms and conditions listed here.
```
' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_SETCURSEL = &H14E

' Select the second item (index 1) available in the list box portion of the combo box control Combo1.
Dim retval As Long ' return value of message
' Change the current selection of Combo1.
retval = SendMessage(Combo1.hWnd, CB_SETCURSEL, ByVal CLng(1), ByVal CLng(0))

See Also

CB_GETCURSEL

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/cb_setcursel.html
CB_SHOWDROPPDOWN Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **CB_SHOWDROPPDOWN** message to a combo box opens or closes its drop-down list box section. Naturally, this message only works if the combo box has a list that can be dropped down, instead of always or never displaying it.

Return Value

This message always returns a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

- *wParam*
  
  If zero, closes the drop-down list of the combo box. If non-zero, opens the drop-down list of the combo box.

- *lParam*
  
  Not used -- set to 0.

Constant Definitions

Const CB_SHOWDROPPDOWN = &H14F

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
Windows API Guide: CB_SHOWDROPPDOWN Message

' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const CB_SHOWDROPPDOWN = &H14F

' Display the drop-down list box of combo box Combo1.
Dim retval As Long ' return value
retval = SendMessage(Combo1.hWnd, CB_SHOWDROPPDOWN, ByVal CLng(1), ByVal CLng(0))

See Also

CB_GETDROPPEDSTATE

Category

Combo Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/c/cb_showdropdown.html
EM_CANUNDO Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the EM_CANUNDO message to an edit control determines if the last operation can be undone. If an undo is allowed, the user can select "Undo" from the control's context menu, and the EM_UNDO message can also be sent to perform an undo.

Return Value

If an undo operation is valid, the function returns a nonzero value. If no undo is possible, the function returns 0.

Visual Basic-Specific Issues

None.

Parameters

wParam
    Not used -- set to 0.

lParam
    Not used -- set to 0.

Constant Definitions

Const EM_CANUNDO = &HC6

Example

Demonstrate undo operations for an edit control. Place three controls on a form window: an edit control (text box) named txtBox, a command button named cmdUndo, and another command button named cmdSet. Pressing cmdSet sets the contents
of txtBox to a predetermined string, a change that cannot be undone. Pressing cmdUndo, naturally, undoes the last change made by the user to txtBox. Whenever the contents of txtBox change, the enabled status of cmdUndo is changed so that the button is enabled if and only if an undo is possible.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_CANUNDO = &HC6
Public Const EM_UNDO = &HC7

' *** Place the following code inside a form window. ***
Private Sub Form_Load()
    ' Make sure cmdUndo is initially disabled.
    cmdUndo.Enabled = False
End Sub

Private Sub txtBox_Change()
    ' Whenever the contents of txtBox change, see if an undo is possible.
    Dim possible As Long  ' is it possible?
    possible = SendMessage(txtBox.hWnd, EM_CANUNDO, ByVal CLng(0), ByVal CLng(0))
    ' Since 0 = False and anything else = True, we can do this:
    cmdUndo.Enabled = possible
End Sub

Private Sub cmdUndo_Click()
    ' Undo the last change the user made to the contents of txtBox.
    Dim retval As Long  ' return value
    retval = SendMessage(txtBox.hWnd, EM_UNDO, ByVal CLng(0), ByVal CLng(0))
    ' Since undo operations can themselves be undone, there's no
    ' reason to send the EM_CANUNDO message here. cmdUndo is already enabled.
End Sub

Private Sub cmdSet_Click()
    ' Set the contents of txtBox to a predetermined string. This can't be undone
    ' using the Undo command or the EM_UNDO message.
    txtBox.Text = "You can't use Undo until you change this text!"
End Sub

See Also

EM_UNDO

Category
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_canundo.html
EM_GETFIRSTVISIBLELINE Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the EM_GETFIRSTVISIBLELINE message to a multi-line edit control finds out which line is the first line visible. This is the line that is currently displayed at the top of the control.

Return Value

If the edit control is multi-line, the message returns the zero-based index of the first visible line at the top of the control. If the edit control is single-line, the message returns the zero-based index of the first visible character.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - Not used -- set to 0.
- lParam
  - Not used -- set to 0.

Constant Definitions

Const EM_GETFIRSTVISIBLELINE = &HCE

Example

Read the first visible line at the top of edit control Text1. Display the text on that line in the Debug window. This requires sending a series of edit control messages, if we do this via the API. To use this example, place a text edit control named Text1
Windows API Guide: EM_GETFIRSTVISIBLELINE Message

and a command button named Command1 on a form window. Make sure that the MultiLine property of Text1 is set to True before running the example. To get the first line of text, click button Command1.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETFIRSTVISIBLELINE = &HCE
Public Const EM_GETLINE = &HC4
Public Const EM_LINEINDEX = &HBB
Public Const EM_LINELENGTH = &HC1

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim lineindex As Long   ' index of the first visible line
    Dim charindex As Long  ' index of the first character on that line
    Dim linetextlen As Integer  ' length of that line (note the data type!)
    Dim linetext As String   ' receives the line's text
    Dim retval As Long      ' generic return value

    ' Get the zero-based index of Text1's first visible line.
    lineindex = SendMessage(Text1.hWnd, EM_GETFIRSTVISIBLELINE, ByVal CLng(0), ByVal CLng(0))

    ' Get the zero-based index of the first character on that line. This is
    ' need for the message we'll send next.
    charindex = SendMessage(Text1.hWnd, EM_LINEINDEX, ByVal lineindex, ByVal CLng(0))

    ' Find out the number of characters on that line. Note how
    ' we store this in a 16-bit value instead of the regular 32-bit Long.
    linetextlen = SendMessage(Text1.hWnd, EM_LINELENGTH, ByVal charindex, ByVal CLng(0))

    ' Make enough room in the string to receive the text. However,
    ' the string must be at least two bytes/characters long for what we'll do
    ' next.
    linetext = Space(IIf(linetextlen >= 2, linetextlen, 2))

    ' EM_GETLINE wants the length of the string copied into the first
    ' two bytes of the string passed to it. Unusual, but this is the simplest
    ' way around
    ' the two-message-parameter limit. This is why linetextlen is only 16 bits
    ' long.
    CopyMemory ByVal linetext, linetextlen, Len(linetextlen)

    ' Finally, read the first line visible in Text1.
    retval = SendMessage(Text1.hWnd, EM_GETLINE, ByVal lineindex, ByVal linetext)

    ' In case we made the string too long (if the line was less than two
' characters long), shorten it back up.
If linetextlen < 2 Then linetext = Left(linetext, linetextlen)
' Finally, display the text.
Debug.Print "The first line visible in Text1 reads: "; linetext
End Sub

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_getfirstvisibleline.html
EM_GETLINE Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **EM_GETLINE** message to an edit control retrieves the contents of one of its lines. The copied text is placed into the string passed as **lParam**.

Return Value

If successful, the message returns the number of characters copied into the string passed as **lParam**. If an error occurred (most likely if an invalid line was specified), the message returns zero.

Visual Basic-Specific Issues

None.

Parameters

**wParam**  
The zero-based index of the line to copy into the string passed as **lParam**.

**lParam**  
A string that receives a copy of the contents of the specified line of the edit control. This string must already be long enough to receive the copied text. Also, before sending the message, the first two bytes of the string must contain a 16-bit integer specifying the length of the string. (See the example for a demonstration of how to do this.)

Constant Definitions

Const **EM_GETLINE** = &HC4

Example
Read the first visible line at the top of edit control Text1. Display the text on that line in the Debug window. This requires sending a series of edit control messages, if we do this via the API. To use this example, place a text edit control named Text1 and a command button named Command1 on a form window. Make sure that the MultiLine property of Text1 is set to True before running the example. To get the first line of text, click button Command1.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source _
As Any, ByVal Length As Long)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal _
Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETFIRSTVISIBLELINE = &HCE
Public Const EM_GETLINE = &HC4
Public Const EM_LINEINDEX = &HBB
Public Const EM_LINELENGTH = &HC1

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim lineindex As Long   ' index of the first visible line
    Dim charindex As Long   ' index of the first character on that line
    Dim linetextlen As Integer  ' length of that line (note the data type!)
    Dim linetext As String    ' receives the line's text
    Dim retval As Long        ' generic return value

    ' Get the zero-based index of Text1's first visible line.
    lineindex = SendMessage(Text1.hWnd, EM_GETFIRSTVISIBLELINE, ByVal CLng(0), ByVal CLng(0))

    ' Get the zero-based index of the first character on that line. This is
    ' need for the message we'll send next.
    charindex = SendMessage(Text1.hWnd, EM_LINEINDEX, ByVal lineindex, ByVal CLng(0))

    ' Find out the number of characters on that line. Note how
    ' we store this in a 16-bit value instead of the regular 32-bit Long.
    linetextlen = SendMessage(Text1.hWnd, EM_LINELENGTH, ByVal charindex, ByVal CLng(0))

    ' Make enough room in the string to receive the text. However,
    ' the string must be at least two bytes/characters long for what we'll do
    ' next.
    linetext = Space(IIf(linetextlen >= 2, linetextlen, 2))

    ' EM_GETLINE wants the length of the string copied into the first
    ' two bytes of the string passed to it. Unusual, but this is the simplest
    ' way around
    ' the two-message-parameter limit. This is why linetextlen is only 16 bits
    ' long.
    CopyMemory ByVal linetext, linetextlen, Len(linetextlen)

    ' Finally, read the first line visible in Text1.
See Also

EM_LINELENGTH

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_getline.html
EM_GETPASSWORDCHAR Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The EM_GETPASSWORDCHAR message identifies the password character used by an edit control. This character appears in place of each character entered into the edit control, in order to hide its contents. As the name implies, this is typically done to hide a password entered into the control.

Return Value

The message returns the ASCII value of the password character used by the edit control. If the control does not use a password character, the message returns 0.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - Not used -- set to 0.
- lParam
  - Not used -- set to 0.

Constant Definitions

Const EM_GETPASSWORDCHAR = &HD2

Example

Identify the password character used by edit control Text1 to hide its contents. To use this example, place a text edit box...
named Text1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd
_  As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETPASSWORDCHAR = &HD2

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim passchar As Long  ' ASCII code of the password character
    passchar = SendMessage(Text1.hWnd, EM_GETPASSWORDCHAR, ByVal CLng(0), ByVal
        CLng(0))
    If passchar > 0 Then
        Debug.Print "The password character is "; Chr(passchar)
    Else
        Debug.Print "Text1 does not use a password character."
    End If
End Sub

See Also
EM_SETPASSWORDCHAR

Category
Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_getpasswordchar.html
EM_GETSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The EM_GETSEL message identifies what text is currently selected in an edit control. The message effectively gives the character positions of the beginning and end of the selection.

Return Value

The message returns the start and end positions of the text selection, packed into a single 32-bit integer. The low-order word contains the starting position of the selection and the high-order word contains the position of the first character after the selection. However, it is easier to use the values placed into wParam and lParam to read the selection instead of using the return value.

Visual Basic-Specific Issues

None.

Parameters

wParam

  Receives the zero-based position of the first character in the selection.

lParam

  Receives the zero-based position of the character immediately after the end of the selection.

Constant Definitions

Const EM_GETSEL = &HB0

Example
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETSEL = &HB0

' Display the selected text in edit control Text1. This is done by sending messages to get the selection, and then accessing the control's .Text property to read its contents.
Dim startpos As Long   ' starting position of the selection
Dim endpos As Long     ' ending position of the selection
Dim retval As Long     ' return value
Dim seltext As String  ' the selected text

' Figure out where the selection in Text1 is.
retval = SendMessage(Text1.hWnd, EM_GETSEL, startpos, endpos)

' Now isolate that selection from the entire text and display it.
If startpos <> endpos Then
    seltext = Mid(Text1.Text, startpos + 1, endpos - startpos)
    Debug.Print "Text selection: "; seltext
Else
    Debug.Print "No selection."
End If

See Also

EM_REPLACESEL, EM_SETSEL

Category

Edit Controls

Last Modified: May 21, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_getsel.html
EM_LINEINDEX Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **EM_LINEINDEX** to an edit control gets the index of the first character on a line. This index is basically the number of characters between that character and the first character in the edit control.

Return Value

The message returns the zero-based index of the first character on the specified line of the edit control. The message returns -1 if **wParam** refers to a line that does not exist.

Visual Basic-Specific Issues

None.

Parameters

**wParam**

The zero-based index of the line. The message will return the index of this line's first character. A value of -1 for this parameter indicates the line that currently contains the caret.

**lParam**

Not used -- set to 0.

Constant Definitions

**Const**  **EM_LINEINDEX**  =  &HBB

Example
Read the first visible line at the top of edit control Text1. Display the text on that line in the Debug window. This requires sending a series of edit control messages, if we do this via the API. To use this example, place a text edit control named Text1 and a command button named Command1 on a form window. Make sure that the MultiLine property of Text1 is set to True before running the example. To get the first line of text, click button Command1.

This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/e/em_lineindex.html).

Declarations and such needed for the example:

(Copy them to the (declarations) section of a module.)

```vba
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETFIRSTVISIBLELINE = &HCE
Public Const EM_GETLINE = &HC4
Public Const EM_LINEINDEX = &HBB
Public Const EM_LINELENGTH = &HC1
```

*** Place the following code inside the form window. ***

```vba
Private Sub Command1_Click()
    Dim lineindex As Long ' index of the first visible line
    Dim charindex As Long ' index of the first character on that line
    Dim linetextlen As Integer ' length of that line (note the data type!)
    Dim linetext As String ' receives the line's text
    Dim retval As Long ' generic return value

    ' Get the zero-based index of Text1's first visible line.
    lineindex = SendMessage(Text1.hWnd, EM_GETFIRSTVISIBLELINE, ByVal CLng(0), ByVal CLng(0))

    ' Get the zero-based index of the first character on that line. This is ' need for the message we'll send next.
    charindex = SendMessage(Text1.hWnd, EM_LINEINDEX, ByVal lineindex, ByVal CLng(0))

    ' Find out the number of characters on that line. Note how ' we store this in a 16-bit value instead of the regular 32-bit Long.
    linetextlen = SendMessage(Text1.hWnd, EM_LINELENGTH, ByVal charindex, ByVal CLng(0))

    ' Make enough room in the string to receive the text. However, ' the string must be at least two bytes/characters long for what we'll do next.
    linetext = Space(IIf(linetextlen >= 2, linetextlen, 2))

    ' EM_GETLINE wants the length of the string copied into the first ' two bytes of the string passed to it. Unusual, but this is the simplest way around ' the two-message-parameter limit. This is why linetextlen is only 16 bits long.
    CopyMemory ByVal linetext, linetextlen, Len(linetextlen)

    ' Finally, read the first line visible in Text1.
```
retval = SendMessage(Text1.hWnd, EM_GETLINE, ByVal lineindex, ByVal linetext)
' In case we made the string too long (if the line was less than two ' characters long), shorten it back up.
If linetextlen < 2 Then linetext = Left(linetext, linetextlen)
' Finally, display the text.
Debug.Print "The first line visible in Text1 reads: "; linetext
End Sub

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_lineindex.html
EM_LINELENGTH Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the EM_LINELENGTH message to an edit control find out how many characters are on a particular line. This message finds out the length of one of the control's lines, taking word wrapping into account. However, for some bizarre reason, the line is identified by a character position and not a line index value.

Return Value

For multi-line edit controls, the message returns the number of characters on the specified line. For single-line edit controls, the message returns the total number of characters in the control.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based index of any character on the line to get the length of. Note that this is a character index, not a line index. (To get the index of the first character on a particular line, send the EM_LINEINDEX message before sending this message.) If this parameter is -1, the function returns the total number of unselected characters that are on lines that contain selected characters.

lParam

Not used -- set to 0.

Constant Definitions

Const EM_LINELENGTH = &HC1
Example

Read the first visible line at the top of edit control Text1. Display the text on that line in the Debug window. This requires sending a series of edit control messages, if we do this via the API. To use this example, place a text edit control named Text1 and a command button named Command1 on a form window. Make sure that the MultiLine property of Text1 is set to True before running the example. To get the first line of text, click button Command1.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal Length As Long)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_GETFIRSTVISIBLELINE = &HCE
Public Const EM_GETLINE = &HC4
Public Const EM_LINEINDEX = &HBB
Public Const EM_LINELENGTH = &HC1

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
Dim lineindex As Long       ' index of the first visible line
Dim charindex As Long       ' index of the first character on that line
Dim linetextlen As Integer  ' length of that line (note the data type!)
Dim linetext As String      ' receives the line's text
Dim retval As Long          ' generic return value

' Get the zero-based index of Text1's first visible line.
lineindex = SendMessage(Text1.hWnd, EM_GETFIRSTVISIBLELINE, ByVal CLng(0), ByVal CLng(0))

' Get the zero-based index of the first character on that line. This is needed for the message we'll send next.
charindex = SendMessage(Text1.hWnd, EM_LINEINDEX, ByVal lineindex, ByVal CLng(0))

' Find out the number of characters on that line. Note how we store this in a 16-bit value instead of the regular 32-bit Long.
linetextlen = SendMessage(Text1.hWnd, EM_LINELENGTH, ByVal charindex, ByVal CLng(0))

' Make enough room in the string to receive the text. However, the string must be at least two bytes/characters long for what we'll do next.
linetext = Space(IIf(linetextlen >= 2, linetextlen, 2))

' EM_GETLINE wants the length of the string copied into the first two bytes of the string passed to it. Unusual, but this is the simplest way around the two-message-parameter limit. This is why linetextlen is only 16 bits long.

CopyMemory ByVal linetext, linetextlen, Len(linetextlen)
' Finally, read the first line visible in Text1.
retval = SendMessage(Text1.hWnd, EM_GETLINE, ByVal lineindex, ByVal linetext)
' In case we made the string too long (if the line was less than two
' characters long), shorten it back up.
If linetextlen < 2 Then linetext = Left(linetext, linetextlen)
' Finally, display the text.
Debug.Print "The first line visible in Text1 reads: "; linetext
End Sub

See Also

EM_GETLINE

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_linelength.html
EM_REPLACESESEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The EM_REPLACESESEL message tells an edit control to replace the currently selected text with a different string. If nothing in the edit control is selected, then the text is instead inserted at the position of the caret.

Return Value

This message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

wParam

Specifies whether the text replacement should be added to the edit control's Undo list, which would allow the user to undo the operation. A nonzero value allows an undo; a value of 0 does not.

lParam

The text to replace the selection with. If nothing in the edit control is selected, this text is inserted at the current position of the caret.

Constant Definitions

Const EM_REPLACESESEL = &HC2

Example
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_REPLACESEL = &HC2
Public Const EM_SETSEL = &HB1

' Select the first five characters in edit control Text1. Then replace the newly
' selected text with the words "REPLACEMENT TEXT".
Dim retval As Long ' return value

' First, select the first five characters of Text1.
retval = SendMessage(Text1.hWnd, EM_SETSEL, ByVal CLng(0), ByVal CLng(5))
' Then replace the selection with some other text. Allow the user to undo it.
retval = SendMessage(Text1.hWnd, EM_REPLACESEL, ByVal CLng(1), ByVal "REPLACEMENT TEXT")

See Also
EM_GETSEL, EM_SETSEL

Category
Edit Controls

Back to the Message list.
Back to the Reference section.
EM_SETPASSWORDCHAR Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The EM_SETPASSWORDCHAR message sets the password character used by an edit control. This character appears in place of each character entered into the edit control, in order to hide its contents. As the name implies, this is typically done to hide a password entered into the control. This message can also be used to tell an edit control not to use such a password character, instead telling it to actually display its contents.

Return Value

EM_SETPASSWORDCHAR does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

wParam

The ASCII code of the character to use as the new password character. If this is 0, then the edit control will no longer use a password character.

lParam

Not used -- set to 0.

Constant Definitions

Const EM_SETPASSWORDCHAR = &HCC

Example
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long

Public Const EM_SETPASSWORDCHAR = &HCC

Dim retval As Long  ' meaningless return value
retval = SendMessage(Text1.hWnd, EM_SETPASSWORDCHAR, ByVal CLng(Asc("-")), ByVal CLng(0))
EM_SETSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The EM_SETSEL message changes the current text selection inside an edit control. Once this message sets the selection, the edit control's caret is placed immediately at the end of the selection.

Return Value

This message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based position of the first character in the selection. If this is -1, no text is selected.

lParam

The zero-based position of the first character that appears immediately after the end of the selection. In other words, this is one plus the position of the last selected character. If this is -1, the selection extends to the end of the control.

Constant Definitions

Const EM_SETSEL = &H1

Example

' This code is licensed according to the terms and conditions listed here.
Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_REPLACESEL = &HC2
Public Const EM_SETSEL = &HB1

Select the first five characters in edit control Text1. Then replace the newly selected text with the words "REPLACEMENT TEXT".
Dim retval As Long ' return value

First, select the first five characters of Text1.
retval = SendMessage(Text1.hWnd, EM_SETSEL, ByVal CLng(0), ByVal CLng(5))
Then replace the selection with some other text. Allow the user to undo it.
retval = SendMessage(Text1.hWnd, EM_REPLACESEL, ByVal CLng(1), ByVal "REPLACEMENT TEXT")

See Also

EM_GETSEL, EM_REPLACESEL

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: May 21, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page. E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_setsel.html
EM_UNDO Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the EM_UNDO message to an edit control undoes the last change made to the control's text. Most changes to an edit control can be undone. However, programmatically setting the text (such as using WM_SETTEXT, for example) cannot be undone. On the other hand, though, undo operations themselves can be undone. The EM_UNDO message has the same effect as selecting "Undo" from the edit control's context menu.

Return Value

If the edit control is a single-line edit control, the message always returns a nonzero value.

If the edit control is a multi-line edit control, the message returns a nonzero value if the undo was successful and zero if the undo failed.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to 0.

lParam
Not used -- set to 0.

Constant Definitions

Const EM_UNDO = &HC7
Example

Demonstrate undo operations for an edit control. Place three controls on a form window: an edit control (text box) named txtBox, a command button named cmdUndo, and another command button named cmdSet. Pressing cmdSet sets the contents of txtBox to a predetermined string, a change that cannot be undone. Pressing cmdUndo, naturally, undoes the last change made by the user to txtBox. Whenever the contents of txtBox change, the enabled status of cmdUndo is changed so that the button is enabled if and only if an undo is possible.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal _
    Msg As Long, wParam As Any, lParam As Any) As Long
Public Const EM_CANUNDO = &HC6
Public Const EM_UNDO = &HC7

' *** Place the following code inside a form window. ***
Private Sub Form_Load()
    ' Make sure cmdUndo is initially disabled.
    cmdUndo.Enabled = False
End Sub

Private Sub txtBox_Change()
    ' Whenever the contents of txtBox change, see if an undo is possible.
    Dim possible As Long  ' is it possible?
    possible = SendMessage(txtBox.hWnd, EM_CANUNDO, ByVal CLng(0), ByVal CLng(0))
    ' Since 0 = False and anything else = True, we can do this:
    cmdUndo.Enabled = possible
End Sub

Private Sub cmdUndo_Click()
    ' Undo the last change the user made to the contents of txtBox.
    Dim retval As Long  ' return value
    retval = SendMessage(txtBox.hWnd, EM_UNDO, ByVal CLng(0), ByVal CLng(0))
    ' Since undo operations can themselves be undone, there's no
    ' reason to send the EM_CANUNDO message here. cmdUndo is already enabled.
End Sub

Private Sub cmdSet_Click()
    ' Set the contents of txtBox to a predetermined string. This can't be undone
    ' using the Undo command or the EM_UNDO message.
    txtBox.Text = "You can't use Undo until you change this text!"
End Sub

See Also
EM_CANUNDO

Category

Edit Controls

Back to the Message list.
Back to the Reference section.

Last Modified: August 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/e/em_undo.html

(IPM_CLEARADDRESS Message

Platforms

- **Windows 95**: Requires Internet Explorer 4.0 or later.
- **Windows 98**: Supported.
- **Windows NT**: Requires Internet Explorer 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **IPM_CLEARADDRESS** message to an IP Address control clears its contents.

Return Value

The message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

- `wParam`  
  Not used -- set to zero.
- `lParam`  
  Not used -- set to zero.

Constant Definitions

```vbnet
Const IPM_CLEARADDRESS = &H464
```

Example

When the form loads, create an IP Address control and place it in the upper-left corner of the window. When the user clicks button cmdClear, the contents of the IP Address control are erased.
To use this example, create a command button named cmdClear and place it on a form window. The IP Address control will be created programmatically when the program starts.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, _
    ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x _
    As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, _
    ByVal hWnd As Long, ByVal hMenu As Long, ByVal hInstance As Long, ByVal lParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Any, ByVal lParam As Any) As Long
Public Const IPM_CLEARADDRESS = &H464

' *** Place the following code inside the form window. ***
Private hIPControl As Long  ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE  ' identifies the control to register
    Dim retval As Long                        ' generic return value
    Dim comctls As INITCOMMONCONTROLSEX_TYPE  ' identifies the control to register

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
        Me.hWnd, 0, App.hInstance, ByVal CLng(0))

End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Clear the contents of the IP Address control when the
' use clicks this button.
Private Sub cmdClear_Click()
    Dim retval As Long
    retval = SendMessage(hIPControl, IPM_CLEARADDRESS, ByVal CLng(0), ByVal CLng(0))
End Sub

See Also

IPM_GETADDRESS, IPM_SETADDRESS

Category

IP Address Control

Back to the Message list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/ipm_clearaddress.html
IPM_GETADDRESS Message

Platforms

- **Windows 95**: Requires Internet Explorer 4.0 or later.
- **Windows 98**: Supported.
- **Windows NT**: Requires Internet Explorer 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the IPM_GETADDRESS message to an IP Address control retrieves the address stored in it. The IP address is returned in host byte order, where the field 0 value (left-most component) is in the highest-order byte and the field 3 value (right-most component) is in the lowest-order byte. For an easy way to extract the components of the IP address, use the FIRST_IPADDRESS, SECOND_IPADDRESS, THIRD_IPADDRESS, and FOURTH_IPADDRESS macros.

Return Value

The message returns the number of blank fields in the IP Address control.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**: Not used -- set to zero.
- **lParam**: Receives the IP address stored in the control, in host byte order. If any of the fields in the control are blank, 0 is used to represent them.

Constant Definitions

Const IPM_GETADDRESS = &H466

Example
Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Windows API Guide: IPM_GETADDRESS Message

Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, ByVal lParam As Any) As Long

Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Any, ByVal lParam As Any) As Long

Public Const IPM_ISBLANK = &H469
Public Const IPM_GETADDRESS = &H466

' *** Place the following code in a form window. ***

Private hIPControl As Long  ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE  ' identifies the control to register
    Dim retval As Long                        ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
                                       Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the address in the IP Address control.
Private Sub cmdGetDomain_Click()
    Dim ipAddress_h As Long   ' the IP address, in host byte order

Dim ipAddress_n As Long    ' the IP address, in network byte order
Dim sockinfo As WSADATA   ' information about the Winsock implementation
Dim pHostinfo As Long     ' pointer to information about the host computer
Dim hostinfo As HOSTENT   ' information about the host computer
Dim domainName As String  ' the primary domain name of the host computer
Dim retval As Long        ' generic return value

' Verify that an IP address was entered.
retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
If retval <> 0 Then
    Debug.Print "No IP address was entered!"
    Exit Sub
End If

' Get the IP address entered by the user and verify that all 
' four fields in the address were entered.
retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
If retval < 4 Then
    Debug.Print "An incomplete IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)

' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()

End Sub

See Also

IPM_CLEARADDRESS, IPM_SETADDRESS
IPM_ISBLANK Function

Platforms

- **Windows 95:** Requires Internet Explorer 4.0 or later.
- **Windows 98:** Supported.
- **Windows NT:** Requires Internet Explorer 4.0 or later.
- **Windows 2000:** Supported.
- **Windows CE:** Requires Windows CE 1.0 or later.

Description & Usage

Sending the `IPM_ISBLANK` message to an IP Address control determines whether the control is blank or not. An IP address control is considered blank if all four fields in it are empty.

Return Value

If the IP Address control is blank, the function returns a nonzero value. If the control is not entirely blank, the function returns zero.

Visual Basic-Specific Issues

None.

Parameters

- `wParam` - Not used -- set to zero.
- `lParam` - Not used -- set to zero.

Constant Definitions

```vbnet
Const IPM_ISBLANK = &H469
```

Example

Create an IP Address control and use it to prompt the user for an IP address. When the user clicks button cmdGetDomain, the
program looks up the first domain name assigned to that address.

To use this example, place a command button named cmdGetDomain on a form window. The IP Address control is created and destroyed by invoking API functions directly and does not need to be placed on the form beforehand.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type WSA_DATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData As WSA_DATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function htonl Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Function gethostbyaddr Lib "wsock32.dll" (addr As Long, ByVal length As Long, ByVal protocol As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal lpString2 As Any) As Long
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA"
(ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hWnd As Long, ByVal lParam As Any) As Long

Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_VISIBLE = &H10000000
Public Const WS_CHILD = &H40000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long

Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
Dim retval As Long ' generic return value

' Register the IP Address control window class.
With comctls
    .dwSize = Len(comctls)
    .dwICC = ICC_INTERNET_CLASSES
End With
retval = InitCommonControlsEx(comctls)

' Create the IP Address control in the corner of the window.
hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_VISIBLE Or WS_CHILD, 0, 0, 125, 20, Me.hWnd, 0, App.hInstance, ByVal CLng(0))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Look up the primary domain name of the host computer identified by the address in the IP Address control.
Private Sub cmdGetDomain_Click()
Dim ipAddress_h As Long ' the IP address, in host byte order
Dim ipAddress_n As Long ' the IP address, in network byte order
Dim sockinfo As WSAData ' information about the Winsock implementation
Dim pHostinfo As Long    ' pointer to information about the host computer
Dim hostinfo As HOSTENT  ' information about the host computer
Dim domainName As String ' the primary domain name of the host computer
Dim retval As Long       ' generic return value

' Verify that an IP address was entered.
retval = SendMessage(hIPControl, IPM_ISBLANK, ByVal CLng(0), ByVal CLng(0))
If retval <> 0 Then
    Debug.Print "No IP address was entered!"
    Exit Sub
End If

' Get the IP address entered by the user and verify that all four fields in the address were entered.
retval = SendMessage(hIPControl, IPM_GETADDRESS, ByVal CLng(0), ipAddress_h)
If retval < 4 Then
    Debug.Print "An incomplete IP address was entered!"
    Exit Sub
End If

' Open up a Winsock v2.2 session.
retval = WSAStartup(&H202, sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Convert the IP address into network byte order.
ipAddress_n = htonl(ipAddress_h)
' Get information about the host computer.
pHostinfo = gethostbyaddr(ipAddress_n, 4, AF_INET)
If pHostInfo = 0 Then
    Debug.Print "Could not find a host with the specified IP address."
Else
    ' Copy the data into the structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    ' Copy the host domain name into a string.
    domainName = Space(lstrlen(hostinfo.h_name))
    retval = lstrcpy(domainName, hostinfo.h_name)
    Debug.Print "Domain name is: "; domainName
End If

' End the Winsock session.
retval = WSACleanup()
IPM_SETADDRESS Message

Platforms

- Windows 95: Requires Internet Explorer 4.0 or later.
- Windows 98: Supported.
- Windows NT: Requires Internet Explorer 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the IPM_SETADDRESS message to an IP Address control sets the IP address stored in it.

Return Value

The message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to zero.

lParam
The IP address to put into the IP Address control. The IP address must be packed into a 32-bit integer and stored in host byte order. The MAKEIPADDRESS macro can be used to convert an IP address into this format.

Constant Definitions

Const IPM_SETADDRESS = &H465

Example

When the form window opens, create an IP address control in the upper-left corner and initialize it to the IP address of the computer. No special effort is needed to run this example, since the IP Address control is created programmatically when the
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Type WSADATA
    wVersion As Integer
    wHighVersion As Integer
    szDescription As String * 257
    szSystemStatus As String * 129
    iMaxSockets As Long
    iMaxUdpDg As Long
    lpVendorInfo As Long
End Type
Public Declare Function WSAStartup Lib "wsock32.dll" (ByVal wVersionRequested As Integer, lpWSAData _
    As WSADATA) As Long
Public Declare Function WSACleanup Lib "wsock32.dll" () As Long
Public Type HOSTENT
    h_name As Long
    h_aliases As Long
    h_addrtype As Integer
    h_length As Integer
    h_addr_list As Long
End Type
Public Const AF_INET = 2
Public Declare Function gethostname Lib "wsock32.dll" (ByVal name As String, ByVal namelen As Long) As Long
Public Declare Function gethostbyname Lib "wsock32.dll" (ByVal name As String) As Long
Public Declare Function ntohl Lib "wsock32.dll" (ByVal hostlong As Long) As Long
Public Declare Sub CopyMemory Lib "kernel32.dll" Alias "RtlMoveMemory" (Destination As Any, Source _
    As Any, ByVal length As Long)
Public Declare Function lstrlen Lib "kernel32.dll" Alias "lstrlenA" (ByVal lpString As Any) As Long
Public Declare Function lstrcpy Lib "kernel32.dll" Alias "lstrcpyA" (ByVal lpString1 As Any, ByVal _
    lpString2 As Any) As Long
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As _
    INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, _
    ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, _
    ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, _
    ByVal hWndParent As Long, ByVal lpCreateParams As Any, ByVal hWndMenuAsLong As Long, ByVal _
    lpfnWndProc As Any) As Long
Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hWnd As Long, ByVal hMenu As Long, ByVal hWndClass As Long, ByVal hWndExClass As Long, ByVal hWndChild As Long, ByVal hInstance As Long, lpParam As Any) As Long

Public Const WC_IPADDRESS = "SysIPAddress32"

Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000

Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long

Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long

Public Const IPM_SETADDRESS = &H465

' Define a relevant API macro.

Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

' *** Place the following code inside the form window. ***

Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the
' upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
    Dim sockinfo As WSADATA ' information about Winsock
    Dim hostinfo As HOSTDATA ' information about an Internet host
    Dim pHostinfo As Long ' pointer to a HOSTENT structure
    Dim localhostName As String ' the computer's domain name
    Dim pIPAddress As Long ' pointer to an IP address dword
    Dim ipAddress_n As Long ' the IP address in network byte order
    Dim ipAddress_h As Long ' the IP address in host byte order
    Dim retval As Long ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
                               Me.hWnd, 0, App.hInstance, ByVal CLng(0))

    ' Open a new Winsock session (version 2.2).
    retval = WSAStartup(MAKEWORD(2, 2), sockinfo)
If retval <> 0 Then
    Debug.Print "ERROR: Attempt to open Winsock failed: error"; retval
    Exit Sub
End If

' Get the domain name of the computer, or, failing that, a string
' that gethostbyname can handle to give an IP address.
localhostName = Space(256)
retval = gethostname(localhostName, 256)
localhostName = Left(localhostName, InStr(localhostName, vbCrLf) - 1)
' Get information about this computer on the network.
' Get information about the domain specified in txtDomain.
pHostinfo = gethostbyname(localhostName)
If pHostinfo = 0 Then
    Debug.Print "Unable to resolve domain name."
Else
    ' Copy the data into a HOSTENT structure.
    CopyMemory hostinfo, ByVal pHostinfo, Len(hostinfo)
    If hostinfo.h_addrtype <> AF_INET Then
        Debug.Print "A non-IP address was returned."
    Else
        ' Copy the pointer to the first (and probably only) IP
        ' address in the structure.
        CopyMemory pIPAddress, ByVal hostinfo.h_addr_list, 4
        ' Copy the actual IP address.
        CopyMemory ipAddress_n, ByVal pIPAddress, 4
        ' Convert it to host byte order.
        ipAddress_h = ntohl(ipAddress_n)
        ' Set the IP Address control to hold this address.
        retval = SendMessage(hIPControl, IPM_SETADDRESS, _
                             ByVal CLng(0), ByVal ipAddress_h)
    End If
End If
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

See Also
IPM_CLEARADDRESS, IPM_GETADDRESS

Category
Winsock
IPM_SETFOCUS Message

Platforms

- **Windows 95**: Requires Internet Explorer 4.0 or later.
- **Windows 98**: Supported.
- **Windows NT**: Requires Internet Explorer 4.0 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the IPM_SETFOCUS message to an IP Address control gives the control the keyboard focus. The text in the particular field that is given the focus will automatically be selected.

Return Value

The message does not return a meaningful value.

Visual Basic-Specific Issues

None.

Parameters

`wParam`

Specifies the (zero-based) field of the control to give the focus to. If this is greater than three, then the focus is set to the first blank field, or the first one if all fields are filled.

Constant Definitions

```vb
Const IPM_SETFOCUS = &H468
```

Example

When the form loads, create an IP Address control and place it in the upper-left corner of the window. When the user clicks button cmdClear, the contents of the IP Address control are erased and the keyboard focus is given to the first field in the control.
To use this example, create a command button named cmdClear and place it on a form window. The IP Address control will be created programmatically when the program starts.

'This code is licensed according to the terms and conditions listed here.

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hMenu As Long, ByVal hInstance As Long, lpParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const IPM_CLEARADDRESS = &H464
Public Const IPM_SETFOCUS = &H468

' *** Place the following code inside the form window. ***
Private hIPControl As Long ' handle to the IP Address control

' When the form is initialized, create an IP Address control in the upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
    Dim retval As Long ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With
    retval = InitCommonControlsEx(comctls)

    ' Create the IP Address control in the corner of the window.
    hIPControl = CreateWindowEx(0, WC_IPADDRESS, ",", WS_CHILD Or WS_VISIBLE, 0,
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

' Clear the contents of the IP Address control when the
' use clicks this button.
Private Sub cmdClear_Click()
    Dim retval As Long

    ' Clear the contents of the control.
    retval = SendMessage(hIPControl, IPM_CLEARADDRESS, ByVal CLng(0), ByVal CLng(0))

    ' Give the first field (field 0) the keyboard focus.
    retval = SendMessage(hIPControl, IPM_SETFOCUS, ByVal CLng(0), ByVal CLng(0))
End Sub

Category

IP Address Control

Back to the Message list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/ipm_setfocus.html
IPM_SETRANGE Message

Platforms

- Windows 95: Requires Internet Explorer 4.0 or later.
- Windows 98: Supported.
- Windows NT: Requires Internet Explorer 4.0 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the IPM_SETRANGE method to an IP Address control sets the valid input range. The range restricts which IP address are allowed to be entered in the control. If an out-of-range address is entered, the offending fields are adjusted to fall within range. (To clear the range, set the range of all four fields to 0-255, which is the range for all IP addresses.)

Return Value

If successful, the message returns a non-zero value. If an error occurred, the message returns zero.

Visual Basic-Specific Issues

None.

Parameters

wParam

The (zero-based) field to set the range of.

lParam

A 16-bit integer that has the lower bound of the range in the low-order byte and the upper bound in the high-order byte. Use the MAKEIPRANGE macro to easily create this value.

Constant Definitions

Const IPM_SETRANGE = &H467

Example
When the form loads, create an IP Address control and place it in the upper-left corner of the window. Restrict entries to the range 128.10.0.0 to 128.11.255.255. It is not necessary to prepare any controls for this example, since the IP Address control is created programmatically when the form loads.

'This code is licensed according to the terms and conditions listed [here].

'Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Type INITCOMMONCONTROLSEX_TYPE
    dwSize As Long
    dwICC As Long
End Type
Public Declare Function InitCommonControlsEx Lib "comctl32.dll" (lpInitCtrls As INITCOMMONCONTROLSEX_TYPE) As Long
Public Const ICC_INTERNET_CLASSES = &H800
Public Declare Function CreateWindowEx Lib "user32.dll" Alias "CreateWindowExA" (ByVal dwExStyle As Long, _
    ByVal lpClassName As String, ByVal lpWindowName As String, ByVal dwStyle As Long, ByVal x As Long, ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hWndParent As Long, ByVal hWndAs Long, ByVal hMenu As Long, ByVal hInstance As Long, ByVal lpParam As Any) As Long
Public Const WC_IPADDRESS = "SysIPAddress32"
Public Const WS_CHILD = &H40000000
Public Const WS_VISIBLE = &H10000000
Public Declare Function DestroyWindow Lib "user32.dll" (ByVal hWnd As Long) As Long
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Any, ByVal lParam As Any) As Long
Public Const IPM_SETRANGE = &H467

' Define a useful API macro.
Public Function MAKEIPRANGE (ByVal low As Byte, ByVal high As Byte) As Long
    MAKEIPRANGE = low + CLng(high) * &H100
End Function

' *** Place the following code inside the form window. ***
Private hIPControl As Long ' handle to the IP Address control
' When the form is initialized, create an IP Address control in the
' upper-left corner of the form.
Private Sub Form_Initialize()
    Dim comctls As INITCOMMONCONTROLSEX_TYPE ' identifies the control to register
    Dim retval As Long ' generic return value

    ' Register the IP Address control window class.
    With comctls
        .dwSize = Len(comctls)
        .dwICC = ICC_INTERNET_CLASSES
    End With

    ' Register the control.
    InitCommonControlsEx comctls

    ' Create the IP Address control window.
    hIPControl = CreateWindowEx(WS_CHILD Or WS_VISIBLE, WC_IPADDRESS, _
        "SysIPAddress", WS_CHILD Or WS_VISIBLE, 0, 0, 200, 200, hWnd, hWnd, hInstance, Nothing)
    If hIPControl = 0 Then Exit Sub

    ' Adjust the size to the form.
    Dim tv As Long
    tv = SendMessage(hWnd, IPM_SETRANGE, 0, MAKEIPRANGE(128, 10))
    SendMessage(hWnd, IPM_SETRANGE, 0, MAKEIPRANGE(128, 11))
    tv = SendMessage(hWnd, IPM_SETRANGE, MAKEIPRANGE(128, 10), MAKEIPRANGE(128, 11))
    SendMessage(hWnd, IPM_SETRANGE, MAKEIPRANGE(128, 10), MAKEIPRANGE(128, 11))

    ' Add a message handler for the control.
    AddHandler hIPControl, Event IPM_SetIPRange
End Sub

retval = InitCommonControlsEx(comctls)

' Create the IP Address control in the corner of the window.
hIPControl = CreateWindowEx(0, WC_IPADDRESS, "", WS_CHILD Or WS_VISIBLE, 0, 0, 125, 20, _
               Me.hWnd, 0, App.hInstance, ByVal CLng(0))

' Restrict the range of valid IP address to 128.10.0.0 to 128.11.255.255.
' (We only have to set the first two fields' range, since 0-255 is the default.)
retval = SendMessage(hIPControl, IPM_SETRANGE, ByVal CLng(0), ByVal MAKEIPRANGE(128, 128))
retval = SendMessage(hIPControl, IPM_SETRANGE, ByVal CLng(1), ByVal MAKEIPRANGE(10, 11))
End Sub

' Destroy the IP Address control when the form closes.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    retval = DestroyWindow(hIPControl)
End Sub

Category

IP Address Control

Back to the Message list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/i/ipm_setrange.html
LB_ADDSTRING Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the `LB_ADDSTRING` message to a list box control adds a string to its list. If the list box is sorted, the new string is added in its proper position according to the sort. If the list box is not sorted, the new string is added to the end. To control where the string is added, use the `LB_INSERTSTRING` message instead.

Return Value

If successful, the message returns the zero-based index of the newly added string's position in the list box. If there is insufficient space to store the new string, the message returns `LB_ERRSTRING`. If some other error occurs, the message returns `LB_ERR`.

Visual Basic-Specific Issues

None.

Parameters

- `wParam`:
  - Not used -- set to 0.
- `lParam`:
  - The string to add to the list box.

Constant Definitions

```
Const LB_ADDSTRING = &H180
Const LB_ERR = -1
Const LB_ERRSPACE = -2
```
Example

When the user clicks button Command1, empty the list box List1, and then add three strings to it. The order in which the strings appear will depend on whether the list box is sorted or not.

To use this example, place a list box named List1 and a command button named Command1 on a form window. To verify that the list box’s list is being emptied, you may wish to add some items to it before running the example.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_ADDSTRING = &H180
Public Const LB_RESETCONTENT = &H184

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim retval As Long ' return value

    ' Empty the list box List1.
    retval = SendMessage(List1.hWnd, LB_RESETCONTENT, ByVal CLng(0), ByVal CLng(0))

    ' Now add three strings to List. Their exact placement will depend on whether List1 is sorted or not.
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "First Item Added")
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "Second Item Added")
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "Last Item Added")
End Sub

See Also
LB_DELETESTRING, LB_INSERTSTRING, LB_RESETCONTENT

Category
List Boxes

Back to the Message list.
Back to the Reference section.
LB_DELETESTRING Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_DELETESTRING message to a list box removes one of the items in it.

Return Value

If successful, the message returns the number of items remaining in the list box. If an error occurred, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based index of the string to delete from the list box.

lParam

Not used -- set to zero.

Constant Definitions

Const LB_DELETESTRING = &H182
Const LB_ERR = -1

Example

When the user clicks button Command1, remove the second item from list box List1. To run this example, place a list box named List1 and a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
'(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_DELETESTRING = &H182
Public Const LB_ERR = -1

*** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim result As Long  ' result of string deletion attempt
    ' Remove the second item from List1 and display the result.
    result = SendMessage(List1.hWnd, LB_DELETESTRING, ByVal CLng(1), ByVal CLng(0))
    If result = LB_ERR Then
        Debug.Print "Unable to delete the second string in List1."
    Else
        Debug.Print "There are"; result; "strings left in List1."
    End If
End Sub

See Also

LB_ADDSTRING, LB_INSERTSTRING, LB_RESETCONTENT

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lb_deletestring.html
LB_GETCOUNT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_GETCOUNT message determines how many items exist in a list box. Keep in mind that the list box items are zero-based; the first item has an index of zero, and the last one has an index of the count minus one.

Return Value

If successful, the message returns the number of items that are in the list box control. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to 0.

lParam
Not used -- set to 0.

Constant Definitions

Const LB_GETCOUNT = &H18B

Example

Display the text of the second-to-last item in list box List1 when the user clicks button Command1. Obviously, to use this example you must place a list box named List1 and a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed [here.](#)

Declarations and such needed for the example:

(Copy them to the (declarations) section of a module.)

Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long

Public Const LB_GETCOUNT = &H18B
Public Const LB_GETTEXT = &H189
Public Const LB_GETTEXTLEN = &H18A

*** Place the following code inside a form window. ***

Private Sub Command1_Click()
    Dim count As Long ' number of items in the list box
    Dim s2l As Long   ' index of the second-to-last item
    Dim itemtext As String ' text of that item
    Dim textlen As Long  ' length of the item text

    ' Figure out the index of the second-to-last item by subtracting two from ' the total item count (remember, the index is zero-based).
    count = SendMessage(List1.hWnd, LB_GETCOUNT, ByVal CLng(0), ByVal CLng(0))
    s2l = count - 2

    ' Make the string long enough to receive that item's text.
    textlen = SendMessage(List1.hWnd, LB_GETTEXTLEN, ByVal s2l, ByVal CLng(0))
    itemtext = Space(textlen) & vbNullChar
    ' Get the item text and remove the trailing null.
    textlen = SendMessage(List1.hWnd, LB_GETTEXT, ByVal s2l, ByVal itemtext)
    itemtext = Left(itemtext, textlen)
    ' Finally, display the result.
    Debug.Print "The second-to-last item is "; itemtext
End Sub

Category

List Boxes

Back to the Message list.
Back to the Reference section.
LB_GETCURSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_GETCURSEL message to a single-selection list box gets the index of the selected item. To get the items selected in a multiple-selection list box, send the LB_GETSELITEMS message instead.

Return Value

If successful, the message returns the zero-based index of the selected item. If no item is selected, or if an error occurred, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to zero.

lParam
Not used -- set to zero.

Constant Definitions

Const LB_GETCURSEL = &H188
Const LB_ERR = -1

Example
Print the text of the selected item in single-selection list box List1. To use this example, place a list box named List1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETCURSEL = &H188
Public Const LB_GETTEXT = &H189
Public Const LB_GETTEXTLEN = &H18A
Public Const LB_ERR = -1

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim selitem As Long ' index of the selected item
    Dim textlen As Long ' length of the item's text
    Dim itemtext As String ' text of the selected item

    ' Get the index of the selected item.
    selitem = SendMessage(List1.hWnd, LB_GETCURSEL, ByVal CLng(0), ByVal CLng(0))
    If selitem = LB_ERR Then
        Debug.Print "No item is selected."
    Else
        ' Get the text of the selected item.
        textlen = SendMessage(List1.hWnd, LB_GETTEXTLEN, ByVal selitem, ByVal CLng(0)) + 1
        itemtext = Space(textlen)
        textlen = SendMessage(List1.hWnd, LB_GETTEXT, ByVal selitem, ByVal itemtext)
        ' Print the result.
        Debug.Print "Selected item: "; Left(itemtext, textlen)
    End If
End Sub

See Also

LB_GETSELITEMS, LB_SETCURSEL

Category

List Boxes

Back to the Message list,
Back to the Reference section.
LB_GETSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_GETSEL message to a list box checks to see if a particular item is selected.

Return Value

If the item is selected, the message returns a value greater than zero. If the item is not selected, the message returns zero. If an error occurred, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based index of the item to check the selection state of.

lParam

Not used -- set to zero.

Constant Definitions

Const LB_GETSEL = &H187
Const LB_ERR = -1

Example

Check to see if the third item of list box List1 is selected. To use this example, place a list box named List1 and a command button named Command1 on a form window.
This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETSEL = &H187

 *** Place the following code inside the form window. ***
Private Sub Command1_Click()
    Dim sel As Long  ' selection state of the item
    ' Check the selection state of List1's third item.
    sel = SendMessage(List1.hWnd, LB_GETSEL, ByVal CLng(2), ByVal CLng(0))
    If sel > 0 Then
        Debug.Print "The third item is selected."
    ElseIf sel = 0 Then
        Debug.Print "The third item is not selected."
    Else
        Debug.Print "An error occured!"
    End If
End Sub

See Also

LB_GETCURSEL, LB_GETSELITEMS, LB_SETSEL

Category

List Boxes

Go back to the Message list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lb_getsel.html
LB_GETSELCOUNT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_GETSELCOUNT message to a multiple-selection message box counts the number of items that are selected.

Return Value

If successful, the message returns the number of items that are selected in the list box. If an error occurred, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - Not used -- set to zero.
- lParam
  - Not used -- set to zero.

Constant Definitions

Const LB_GETSELCOUNT = &H190
Const LB_ERR = -1

Example

Display a list of all items that are selected in a list box. To use this example, place a multi-select list box named List1 and a command button named Command1 on a form window.
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETSELCOUNT = &H190
Public Const LB_GETSELITEMS = &H191

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim items() As Long ' indexes of the selected items
    Dim numsel As Long ' number of selected items
    Dim c As Long ' counter variable
    Dim retval As Long ' return value

    ' Count the number of selected items.
    numsel = SendMessage(List1.hWnd, LB_GETSELCOUNT, ByVal CLng(0), ByVal CLng(0))
    If numsel = 0 Then
        Debug.Print "No items are selected."
    Else

        ' Resize the array so it can hold all the indexes.
        ReDim items(0 To numsel - 1) As Long
        ' Get the indexes of all selected items.
        retval = SendMessage(List1.hWnd, LB_GETSELITEMS, ByVal numsel, items(0))
        ' Display them.
        Debug.Print "The following items are selected (identified by index):
        For c = 0 To numsel - 1
            Debug.Print items(c);
        Next c
        Debug.Print
    End If
End Sub

See Also

LB_GETSELITEMS

Category

List Boxes

Back to the Message list.
Back to the Reference section.
LB_GETSELITEMS Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **LB_GETSELITEMS** message to a multiple-selection list box retrieves an list of all the selected items. The index of each selected item is copied into an array passed with the message. For single-selection list boxes, send the **LB_GETCURSEL** message instead.

Return Value

If successful, the message returns the number of item indexes copied into the array. If an error occurred, the function returns **LB_ERR**.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**: The number of list box item indexes that the array passed as **lParam** can hold.
- **lParam**: The array that receives the indexes of the selected items. If the array is too small, then only some of the indexes are copied over. To get the minimum length needed for all the indexes, use the **LB_GETSELCOUNT** message.

Constant Definitions

- `Const LB_GETSELITEMS = &H191`
- `Const LB_ERR = -1`

Example

Display a list of all items that are selected in a list box. To use this example, place a multi-select list box named List1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETSELCOUNT = &H190
Public Const LB_GETSELITEMS = &H191

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim items() As Long  ' indexes of the selected items
    Dim numsel As Long   ' number of selected items
    Dim c As Long        ' counter variable
    Dim retval As Long   ' return value

    ' Count the number of selected items.
    numsel = SendMessage(List1.hWnd, LB_GETSELCOUNT, ByVal CLng(0), ByVal CLng(0))
    If numsel = 0 Then
        Debug.Print "No items are selected."
    Else
        ' Resize the array so it can hold all the indexes.
        ReDim items(0 To numsel - 1) As Long
        ' Get the indexes of all selected items.
        retval = SendMessage(List1.hWnd, LB_GETSELITEMS, ByVal numsel, items(0))
        ' Display them.
        Debug.Print "The following items are selected (identified by index):"
        For c = 0 To numsel - 1
            Debug.Print items(c);
        Next c
    End If
End Sub

See Also

LB_GETCURSEL, LB_GETSEL, LB_GETSELCOUNT

Category

List Boxes
LB_GETTEXT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The LB_GETTEXT message retrieves the text of one of the items in a list box control.

Return Value

If successful, the message returns the length of the string copied into the string passed as lParam, not including the terminating null character. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

wParam

The zero-based index of the list box item to retrieve the text of. (The first item has an index of 0, etc.)

lParam

The string to copy the list box item text into. The string must have enough room to receive the entire string along with a terminating null character.

Constant Definitions

Const LB_GETTEXT = &H189

Example

Display the text of the second-to-last item in list box List1 when the user clicks button Command1. Obviously, to use this example you must place a list box named List1 and a command button named Command1 on a form window.
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETCOUNT = &H18B
Public Const LB_GETTEXT = &H189
Public Const LB_GETTEXTLEN = &H18A

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim count As Long       ' number of items in the list box
    Dim s2l As Long         ' index of the second-to-last item
    Dim itemtext As String  ' text of that item
    Dim textlen As Long     ' length of the item text

    ' Figure out the index of the second-to-last item by subtracting two from
    ' the total item count (remember, the index is zero-based).
    count = SendMessage(List1.hWnd, LB_GETCOUNT, ByVal CLng(0), ByVal CLng(0))
    s2l = count - 2

    ' Make the string long enough to receive that item's text.
    textlen = SendMessage(List1.hWnd, LB_GETTEXTLEN, ByVal s2l, ByVal CLng(0))
    itemtext = Space(textlen) & vbNullChar

    ' Get the item text and remove the trailing null.
    textlen = SendMessage(List1.hWnd, LB_GETTEXT, ByVal s2l, ByVal itemtext)
    itemtext = Left(itemtext, textlen)

    ' Finally, display the result.
    Debug.Print "The second-to-last item is "; itemtext
End Sub

See Also

LB_GETTEXTLEN

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
LB_GETTEXTLEN Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The **LB_GETTEXTLEN** message retrieves the length of the text of one of the items in a list box control. The reported length does not include its terminating null character. When determining how long to size a string to receive the list box item's text, remember to add one to include the null.

Return Value

If successful, the message returns the length of the list box item's text, not including the terminating null character. If an error occurred, the message returns -1.

Visual Basic-Specific Issues

None.

Parameters

`wParam`

The zero-based index of the list box item to retrieve the length of the text of. (The first item has an index of 0, etc.)

`lParam`

Not used -- set to 0.

Constant Definitions

```vb
Const LB_GETTEXTLEN = &H18A
```

Example

Display the text of the second-to-last item in list box List1 when the user clicks button Command1. Obviously, to use this
example you must place a list box named List1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/l/lb_gettextlen.html).  

' Declarations and such needed for the example:  
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_GETCOUNT = &H18B
Public Const LB_GETTEXT = &H189
Public Const LB_GETTEXTLEN = &H18A

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim count As Long       ' number of items in the list box
    Dim s2l As Long         ' index of the second-to-last item
    Dim itemtext As String  ' text of that item
    Dim textlen As Long     ' length of the item text

    ' Figure out the index of the second-to-last item by subtracting two from  
    ' the total item count (remember, the index is zero-based).
    count = SendMessage(List1.hWnd, LB_GETCOUNT, ByVal CLng(0), ByVal CLng(0))
    s2l = count - 2

    ' Make the string long enough to receive that item's text.
    textlen = SendMessage(List1.hWnd, LB_GETTEXTLEN, ByVal s2l, ByVal CLng(0))
    itemtext = Space(textlen) & vbNullChar

    ' Get the item text and remove the trailing null.
    textlen = SendMessage(List1.hWnd, LB_GETTEXT, ByVal s2l, ByVal itemtext)
    itemtext = Left(itemtext, textlen)

    ' Finally, display the result.
    Debug.Print "The second-to-last item is "; itemtext
End Sub

See Also


Category

LB_INSERTSTRING Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Sending the **LB_INSERTSTRING** message to a list box inserts a string into it. The string is placed at the position specified in the parameters, regardless of whether the list box is sorted or not.

Return Value

If successful, the message returns the zero-based index of the newly added string's position in the list box. If there is insufficient space to store the new string, the message returns **LB_ERRSPACE**. If some other error occurs, the message returns **LB_ERR**.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  The zero-based index of the position to insert the string in the list box. If this is -1, the string is added to the end of the list.

- **lParam**
  The string to add to the list box.

Constant Definitions

```vbnet
Const LB_INSERTSTRING = &H181
Const LB_ERR = -1
Const LB_ERRSPACE = -2
```
Example

When the user clicks button Command1, insert three strings into list box List1. One string is added to the beginning, one to the third position, and one to the end. To run this example, place a list box named List1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Any, ByVal lParam As Any) As Long
Public Const LB_INSERTSTRING = &H181

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim retval As Long  ' return value
    ' Add a string to the beginning of the drop-down box.
    retval = SendMessage(List1.hWnd, LB_INSERTSTRING, ByVal CLng(0), ByVal "First Item")
    ' Insert a string at the third position in the drop-down box.
    retval = SendMessage(List1.hWnd, LB_INSERTSTRING, ByVal CLng(2), ByVal "Third Item")
    ' Add a string to the end of the drop-down box.
    retval = SendMessage(List1.hWnd, LB_INSERTSTRING, ByVal CLng(-1), ByVal "Last Item")
End Sub

See Also

LB_ADDSTRING, LB_DELETESTRING, LB_RESETCONTENT

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
LB_RESETCONTENT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_RESETCONTENT message to a list box removes all items from it.

Return Value

This message always returns zero.

Visual Basic-Specific Issues

None.

Parameters

wParam
   Not used -- set to 0.

lParam
   Not used -- set to 0.

Constant Definitions

Const LB_RESETCONTENT = &H184

Example

When the user clicks button Command1, empty the list box List1, and then add three strings to it. The order in which the strings appear will depend on whether the list box is sorted or not.
To use this example, place a list box named List1 and a command button named Command1 on a form window. To verify that the list box's list is being emptied, you may wish to add some items to it before running the example.

'This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_ADDSTRING = &H180
Public Const LB_RESETCONTENT = &H184

' *** Place the following code inside a form window. ***
Private Sub Command1_Click()
    Dim retval As Long  ' return value

    ' Empty the list box List1.
    retval = SendMessage(List1.hWnd, LB_RESETCONTENT, ByVal CLng(0), ByVal CLng(0))

    ' Now add three strings to List.  Their exact placement will depend on whether List1 is sorted or not.
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "First Item Added")
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "Second Item Added")
    retval = SendMessage(List1.hWnd, LB_ADDSTRING, ByVal CLng(0), ByVal "Last Item Added")
End Sub

See Also

LB_ADDSTRING, LB_DELETESTRING, LB_INSERTSTRING

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
LB_SETCURSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_SETCURSEL message to a single-selection list box sets the item that is currently selected, or clears the selection entirely. For multiple-selection list boxes, send the LB_SETSEL message instead.

Return Value

If successful, the message returns 1. If an error occurred, or if the selection was cleared, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

wParam
The zero-based index of the item to select. To clear the selection, set this to -1.

lParam
Not used -- set to zero.

Constant Definitions

Const LB_SETCURSEL = &H186
Const LB_ERR = -1

Example

Select the second item in list box List1. To use this example, place a single-selection list box named List1 and a command
' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_SETCURSEL = &H186

' *** Place the following code inside the form window. ***
Private Sub Command1_Click()
   Dim retval As Long ' return value
   
   ' Change the selection in list box List1 to the second item.
   retval = SendMessage(List1.hWnd, LB_SETCURSEL, ByVal Clng(1), ByVal Clng(0))
End Sub

See Also

LB_GETCURSEL, LB_SETSEL

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lb_setcursel.html
LB_SETSEL Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the LB_SETSEL to a multiple-selection list box selects or deselects one of its items. Alternatively, the message can select or deselect all of the items simultaneously. For a single-selection list box, send the LB_SETCURSEL message instead.

Return Value

If successful, the message returns zero. If an error occurred, the message returns LB_ERR.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - If this is zero, the item is deselected. If this is a non-zero value, the item is selected.
- lParam
  - The zero-based index of the item to select or deselect. If this is -1, all items in the list box are selected or deselected.

Constant Definitions

Const LB_SETSEL = &H185
Const LB_ERR = -1

Example

Change the selection in list box List1 so that the only selected items are the first and third items. To use this example, place a
multiple-selection list box named List1 and a command button named Command1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, _
    ByVal Msg As Long, wParam As Any, lParam As Any) As Long
Public Const LB_SETSEL = &H185

' *** Place the following code inside the form window. ***

Private Sub Command1_Click()
    Dim retval As Long  ' return value

    ' First, deselect all items in the list box.
    retval = SendMessage(List1.hWnd, LB_SETSEL, ByVal CLng(0), ByVal CLng(-1))
    ' Then, select the first and third items.
    retval = SendMessage(List1.hWnd, LB_SETSEL, ByVal CLng(1), ByVal CLng(0))
    retval = SendMessage(List1.hWnd, LB_SETSEL, ByVal CLng(1), ByVal CLng(2))
End Sub

See Also

LB_GETSEL, LB_SETCURSEL

Category

List Boxes

Back to the Message list.
Back to the Reference section.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/l/lb_setssel.html
MM_MCINOTIFY Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

A window receives the **MM_MCINOTIFY** message in response to the completion of a MCI device's action when the "notify" option is used. This tells the window that the MCI command has finished, whether because of success, failure, or some other event. The **MM_MCINOTIFY** message is only sent when the "notify" option was originally specified.

Return Value

The **MM_MCINOTIFY** message should return 0.

Visual Basic-Specific Issues

None.

Parameters

**wParam**

One of the following flags specifying the result of the original MCI command:

- **MCI_NOTIFY_ABORTED**
  
  The MCI device aborted execution of the command.

- **MCI_NOTIFY_FAILURE**
  
  An error occurred while executing the command.

- **MCI_NOTIFY_SUCCESSFUL**
  
  The MCI device successfully executed the command.

- **MCI_NOTIFY_SUPERSEDED**
  
  Another command for the MCI device requested notification, so the window will not receive any notification when the command actually finishes.

**lParam**

The identifier of the MCI device that sent the message.
Constant Definitions

Const MM_MCINOTIFY = &H3B9
Const MCI_NOTIFY_ABORTED = &H4
Const MCI_NOTIFY_FAILURE = &H8
Const MCI_NOTIFY_SUCCESSFUL = &H1
Const MCI_NOTIFY_SUPERSEDED = &H2

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
lpstrCommand As String, ByVal lpstrReturnString As String, ByVal uReturnLength _
As Long, ByVal hwndCallback As Long) As Long
Public Const MM_MCINOTIFY = &H3B9
Public Const MCI_NOTIFY_ABORTED = &H4
Public Const MCI_NOTIFY_FAILURE = &H8
Public Const MCI_NOTIFY_SUCCESSFUL = &H1
Public Const MCI_NOTIFY_SUPERSEDED = &H2
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal _
lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As _
long, ByVal lParam As Long) As Long
Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal _
hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = (-4)

' When the user pushes button Command1, begin playing a MIDI file. Do not wait
' for the music to finish, but display a simple message prompt when the end of
' the MIDI file has been reached. Do this by using the "notify" option of the "play"
' MCI command. Make sure the device is closed before quitting.

' *** Place the following code inside a module. ***

' Pointer to window Form1's previous window procedure.
Public pOldProc As Long

' Custom window procedure for Form1.
Public Function WindowProc(ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As _
long, ByVal lParam As Long) As Long
Dim mbtext As String  ' text of message box
Dim retval As Long    ' return value

' If the notification message is received, tell the user how
' the playback of the MIDI file concluded.
Select Case uMsg
Case MM_MCINOTIFY
    Select Case wParam
    Case MCI_NOTIFY_ABORTED
        mbtext = "Playback of the MIDI file was somehow aborted."
    Case MCI_NOTIFY_FAILURE
        mbtext = "An error occurred while playing the MIDI file."
    Case MCI_NOTIFY_SUCCESSFUL
        mbtext = "Playback of the MIDI file concluded successfully."
    Case MCI_NOTIFY_SUPERSEDED
        mbtext = "Another command requested notification from this device."
    End Select
    retval = MsgBox(mbtext, vbOkOnly Or vbInformation)
End Select
End Function

' *** Place the following code inside Form1. ***

Private Sub Command1_Click()
    ' Open and start playing a MIDI file. Have the device notify Form1 once
    ' playback has ended.
    Dim retval As Long  ' return value
    retval = mciSendString("open C:\Music\song.mid alias music", ",", 0, 0)
    retval = mciSendString("play music notify", ",", 0, Form1.hWnd)
End Sub

Private Sub Form1_Load()
    ' Set up the custom window procedure for use with Form1.
    pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

Private Sub Form1_Unload(Cancel As Integer)
    Dim retval As Long  ' return value
    ' Make sure that the MIDI file is closed.
    retval = mciSendString("close music", ",", 0, 0)
    ' Restore Form1's original window procedure.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
End Sub

Category

Media Control Interface (MCI)

Back to the Function list.
Windows API Guide: MM_MCINOTIFY Message

Back to the Reference section.

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/mm_mcinotify.html

WM_COMMAND Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

A window receives the WM_COMMAND message when the user selects an item in one of the window's menus. This includes any popup menus which the window owns. Upon receiving this message, the window should perform whatever task or operation the menu item was meant to perform.

Return Value

The WM_COMMAND message should always return 0.

Visual Basic-Specific Issues

None.

Parameters

wParam

The high-order word is the notification code of the message: 0 means the message was generated by selecting a menu item, whereas 1 means the message was generated by a keyboard accelerator. The low-order word contains the identifier of the menu item that the user selected. The LOWORD and HIWORD macros can extract these two values.

lParam

A handle to the control that sent this message, if any. If no control sent the message, this is 0.

Constant Definitions

Const WM_COMMAND = &H111

Example
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/w/wm_command.html).

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

```vbscript
Public Declare Function CreatePopupMenu Lib "user32.dll" () As Long
Public Declare Function DestroyMenu Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUTEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_DEFAULT = &H1000
Public Const MFS_ENABLED = &H0
Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal _
    hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmmi As _
    MENUTEMINFO) As Long
Public Type RECT
    left As Long
    top As Long
    right As Long
    bottom As Long
End Type
Public Type TPMPARAMS
    cbSize As Long
    rcExclude As RECT
End Type
Public Declare Function TrackPopupMenuEx Lib "user32.dll" (ByVal hMenu As Long, ByVal fuFlags As Long, ByVal x As Long, ByVal y As Long, ByVal hWnd As Long, lptpm As _
    TPMPARAMS) As Long
Public Const TPM_LEFTALIGN = &H0
Public Const TPM_TOPALIGN = &H0
Public Const TPM_LEFTBUTTON = &H0
Public Type POINT_TYPE
    x As Long
End Type
```

http://216.26.168.92/vbapi/ref/w/wm_command.html (2 of 6) [9/1/2002 6:09:00 PM]
When the user clicks button Command1, have a very simple popup menu appear. The menu only has two options, divided by a separator bar. The menu is created when needed and is destroyed after its use.

*** Place the following code inside a module. ***

The following application-defined constants are used to name the menu item identifiers used by this example. They are not actually part of the API; instead, they are used just to eliminate "magic numbers."

Public Const ID_ABOUT = 101
Public Const ID_SEPARATOR = 102
Public Const ID_EXIT = 103

This is a pointer to Form1's previous window procedure.
Public pOldProc As Long

This is actually the "declaration" of the LOWORD API macro.
Public Function LOWORD (ByVal dwValue As Long) As Integer
    LOWORD = dwValue Mod &H10000
End Function

The following window procedure for window Form1 handles its messages. Specifically, it processes the WM_COMMAND message for the popup menu.
Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim retval As Long  ' return value

Select Case uMsg
Case WM_COMMAND
    ' Do whatever the items on the popup menu requested.
    Select Case LOWORD (wParam)
    Case ID_ABOUT
        ' Use the VB MsgBox function to display a short message in a dialog box. Using the API isn't necessary.
        retval = MsgBox("This example demonstrates how to use the API to " & _
"display a pop-up menu.", vbOkOnly Or vbInformation,

"Windows API Guide")

Case ID_EXIT
    ' End this program by closing and unloading Form1.
   Unload Form1
Case Else
    ' Have the previous window procedure handle other items.
    retval = CallWindowProc(pOldProc, hWnd, uMsg, wParam, lParam)
End Select
Case Else
    WindowProc = CallWindowProc(pOldProc, hWnd, uMsg, wParam, lParam)
End Select
End Function

' *** Place the following code inside Form1. ***

Private Sub Command1_Click()
    Dim hPopupMenu As Long    ' handle to the popup menu to display
    Dim mii As MENUITEMINFO   ' describes menu items to add
    Dim tpm As TPMPARAMS      ' identifies the exclusion rectangle
    Dim curpos As POINT_TYPE  ' holds the current mouse coordinates
    Dim menusel As Long       ' ID of what the user selected in the popup menu
    Dim retval As Long        ' generic return value

    ' Create the popup menu that will be displayed.
    hPopupMenu = CreatePopupMenu()
    ' Add the menu's first item: "About This Problem..."
    With mii
        ' The size of this structure.
        .cbSize = Len(mii)
        ' Which elements of the structure to use.
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        ' The type of item: a string.
        .fType = MFT_STRING
        ' This item is currently enabled and is the default item.
        .fState = MFS_ENABLED Or MFS_DEFAULT
        ' Assign this item an item identifier.
        .wID = ID_ABOUT
        ' Display the following text for the item.
        .dwTypeData = "&About This Example..."
        .cch = Len(.dwTypeData)
    End With
    retval = InsertMenuITEM(hPopupMenu, 0, 1, mii)
    ' Add the second item: a separator bar.
    With mii
        .fType = MFT_SEPARATOR
        .fState = MFS_ENABLED
        .wID = ID_SEPARATOR
    End With
End With
retval = InsertMenuItem(hPopupMenu, 1, 1, mii)
' Add the final item: "Exit".
With mii
  .fType = MFT_STRING
  .wID = ID_EXIT
  .dwTypeData = "E&xit"
  .cch = Len(.dwTypeData)
End With
retval = InsertMenuItem(hPopupMenu, 2, 1, mii)

' Determine where the mouse cursor currently is, in order to have
' the popup menu appear at that point.
retval = GetCursorPos(curpos)

' Make the exclusion rectangle empty because there's no need for it here.
With tpm
  ' Size of the structure.
  .cbSize = Len(tpm)
  ' Make the exclusion rectangle empty.
  retval = SetRectEmpty(.rcExclude)
End With

' Display the popup menu at the mouse cursor. The window procedure
' defined in the module will process the selected item automatically.
menusel = TrackPopupMenuEx(hPopupMenu, TPM_TOPALIGN Or TPM_LEFTALIGN Or _
  TPM_LEFTBUTTON, curpos.x, curpos.y, Form1.hWnd, tpm)

' Destroy the popup menu now.
retval = DestroyMenu(hPopupMenu)
End Sub

' When Form1 loads, tell it to use the custom window procedure.
Private Sub Form_Load()
  ' Set the custom window procedure to process Form1's messages.
  pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' Before unloading, remove the custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
  Dim retval As Long  ' return value

  ' Replace the previous window procedure to prevent crashing.
  retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
End Sub

See Also

WM_SYSCOMMAND

Category
WM_INITMENU Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

A window receives the WM_INITMENU message before one of its menus is displayed. This menu could be either the "regular" menu or the window's system menu. Upon receiving this message, the window should prepare the menu for display. This normally includes making sure any items that should be checked are checked, graying out certain items, etc. A window only receives WM_INITMENU when the user begins looking at a menu; it does not receive the message multiple times as the user looks into submenus.

Return Value

The WM_INITMENU message should always return 0.

Visual Basic-Specific Issues

None.

Parameters

- wParam
  - A handle to the menu which will soon be displayed.
- lParam
  - Not used.

Constant Definitions

Const WM_INITMENU = &H116

Example
'Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long

Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long

Public Type MENUITEMINFO
  cbSize As Long
  fMask As Long
  fType As Long
  fState As Long
  wID As Long
  hSubMenu As Long
  hbmpChecked As Long
  hbmpUnchecked As Long
  dwItemData As Long
  dwTypeData As String
  cch As Long
End Type

Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2
Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8

Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1

Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Public Const GWL_WNDPROC = -4
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProCA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116

' Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.
' Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

' *** Place the following code in a module. ***

Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public ontop As Boolean  ' identifies if Form1 is always on top or not

' The following function acts as Form1's window procedure to process messages.
Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim hSysMenu As Long     ' handle to Form1's system menu
    Dim mii As MENUITEMINFO  ' menu item information for Always On Top
    Dim retval As Long       ' return value
    Select Case uMsg
        Case WM_INITMENU
            ' Before displaying the system menu, make sure that the Always On Top option is properly checked.
            hSysMenu = GetSystemMenu(hwnd, 0)
            With mii
                ' Size of the structure.
                .cbSize = Len(mii)
                ' Only use what needs to be changed.
                .fMask = MIIM_STATE
                ' If Form1 is now always on top, check the item.
                .fState = MFS_ENABLED Or IIf(ontop, MFS_CHECKED, 0)
            End With
            retval = SetMenuItemInfo(hSysMenu, 1, 0, mii)
            WindowProc = 0
        Case WM_SYSCOMMAND
            ' If Always On Top (ID = 1) was selected, change the on top/not on top setting of Form1 to match.
            If wParam = 1 Then
                ' Reverse the setting and make it the current one.
                ontop = Not ontop
                retval = SetWindowPos(hwnd, IIf(ontop, HWND_TOPMOST, HWND_NOTOPMOST), 0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
            Else
                WindowProc = 0
        End Select
    WindowProc = 0
End Function
' Some other item was selected. Let the previous window procedure process it.

WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End If
Case Else
' If this is some other message, let the previous procedure handle it.

WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End Select
End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the new window procedure.
Private Sub Form_Load()
    Dim hSysMenu As Long     ' handle to the system menu
    Dim count As Long        ' the number of items initially on the menu
    Dim mii As MENUITEMINFO  ' describes a menu item to add
    Dim retval As Long       ' return value

    ' Get a handle to the system menu.
hSysMenu = GetSystemMenu(Form1.hWnd, 0)
    ' See how many items are currently in it.
count = GetMenuItemCount(hSysMenu)

    ' Add a separator bar and then Always On Top to the system menu.
    With mii
        ' The size of the structure.
        .cbSize = Len(mii)
        ' What parts of the structure to use.
        .fMask = MIIM_ID Or MIIM_TYPE
        ' This is a separator.
        .fType = MFT_SEPARATOR
        ' It has an ID of 0.
        .wID = 0
    End With
    ' Add the separator to the end of the system menu.
    retval = InsertMenuItem(hSysMenu, count, 1, mii)

    ' Likewise, add the Always On Top command.
    With mii
        .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
        ' This is a regular text item.
        .fType = MFT_STRING
        ' The option is enabled.
        .fState = MFS_ENABLED
        ' It has an ID of 1 (this identifies it in the window procedure).
        .wID = 1
        ' The text to place in the menu item.

.dwTypeData = "&Always On Top"
    .cch = Len(.dwTypeData)

End With
' Add this to the bottom of the system menu.
retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)

' Set the custom window procedure to process Form1's messages.
ontop = False
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)

End Sub

' Before unloading, restore the default system menu and remove the
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
    Dim retval As Long ' return value

    ' Replace the previous window procedure to prevent crashing.
    retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)

    ' Remove the modifications made to the system menu.
    retval = GetSystemMenu(Form1.hWnd, 1)
End Sub

Category

Menus

Back to the Message list.
Back to the Reference section.
WM_SYSCOMMAND Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

A window receives the WM_SYSCOMMAND message when either the user selects an item from its system menu or the user presses the restore, maximize, minimize, or close button on its title bar. Typically, a program should let the default window procedure process this command, unless the program needs to respond to an item added by the program to the window's system menu (see the example below).

Return Value

The WM_SYSCOMMAND message should always return 0.

Visual Basic-Specific Issues

None.

Parameters

wParam

The unique menu item identifier of the selected item. This could be one of the following flags, if the user selected an item that is by default on the system menu. When checking for these flags, first perform a bitwise And &HFFF0 on wParam before checking its value. This is necessary because the lowest four bits are used internally by the system and do not affect the flag itself. Of course, if your program added its own items to the system menu, then wParam will be that item's identifier and not one of the following values.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC_CLOSE</td>
<td>Close the window.</td>
</tr>
<tr>
<td>SC_CONTEXTHELP</td>
<td>Display a context-sensitive help window.</td>
</tr>
<tr>
<td>SC_MAXIMIZE</td>
<td>Maximize the window.</td>
</tr>
<tr>
<td>SC_MINIMIZE</td>
<td></td>
</tr>
</tbody>
</table>
Windows API Guide: WM_SYSCOMMAND Message

Minimize the window.
SC_MOVE
Move the window.
SC_RESTORE
Restore the window.
SC_SIZE
Size the window.

lParam
If the system menu item was selected using the mouse, this contains the coordinates of the mouse cursor. The x-coordinate is stored in the low-order word, and the y-coordinate is stored in the high-order word. The LOWORD and HIWORD macros can be used to extract the coordinates.

Constant Definitions

Const WM_SYSCOMMAND = &H112
Const SC_CLOSE = &HF060
Const SC_CONTEXTHELP = &HF180
Const SC_MAXIMIZE = &HF030
Const SC_MINIMIZE = &HF020
Const SC_MOVE = &HF010
Const SC_RESTORE = &HF120
Const SC_SIZE = &HF000

Example

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
' There's quite a few declarations for this example, but it's worth it!
Public Declare Function GetSystemMenu Lib "user32.dll" (ByVal hWnd As Long, ByVal bRevert As Long) As Long
Public Declare Function GetMenuItemCount Lib "user32.dll" (ByVal hMenu As Long) As Long
Public Type MENUITEMINFO
    cbSize As Long
    fMask As Long
    fType As Long
    fState As Long
    wID As Long
    hSubMenu As Long
    hbmpChecked As Long
    hbmpUnchecked As Long
    dwItemData As Long
    dwTypeData As String
    cch As Long
End Type
Public Const MIIM_STATE = &H1
Public Const MIIM_ID = &H2

Public Const MIIM_TYPE = &H10
Public Const MFT_SEPARATOR = &H800
Public Const MFT_STRING = &H0
Public Const MFS_ENABLED = &H0
Public Const MFS_CHECKED = &H8

Public Declare Function InsertMenuItem Lib "user32.dll" Alias "InsertMenuItemA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetMenuItemInfo Lib "user32.dll" Alias "SetMenuItemInfoA" (ByVal hMenu As Long, ByVal uItem As Long, ByVal fByPosition As Long, lpmii As MENUITEMINFO) As Long

Public Declare Function SetWindowPos Lib "user32.dll" (ByVal hWnd As Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags As Long) As Long

Public Const HWND_TOPMOST = -1
Public Const HWND_NOTOPMOST = -2
Public Const SWP_NOMOVE = &H2
Public Const SWP_NOSIZE = &H1

Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long

Public Const GWL_WNDPROC = -4

Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

Public Const WM_SYSCOMMAND = &H112
Public Const WM_INITMENU = &H116

' Add an option to make window Form1 "Always On Top" to the bottom of its system menu. A check mark appears next to this option when active. The menu item acts as a toggle.
' Note how subclassing the window is necessary to process the two messages needed to give the added system menu item its full functionality.

' *** Place the following code in a module. ***

Public pOldProc As Long  ' pointer to Form1's previous window procedure
Public ontop As Boolean  ' identifies if Form1 is always on top or not

' The following function acts as Form1's window procedure to process messages.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim hSysMenu As Long     ' handle to Form1's system menu
    Dim mii As MENUITEMINFO  ' menu item information for Always On Top
    Dim retval As Long       ' return value

    Select Case uMsg
Case **WM_INITMENU**

' Before displaying the system menu, make sure that the Always On Top 
' option is properly checked.

```
    hSysMenu = GetSystemMenu(hwnd, 0)
    With mii
        ' Size of the structure.
        .cbSize = Len(mii)
        ' Only use what needs to be changed.
        .fMask = MIIM_STATE
        ' If Form1 is now always on top, check the item.
        .fState = MFS_ENABLED Or IIf(ontop, MFS_CHECKED, 0)
    End With
    retval = SetMenuItemInfo(hSysMenu, 1, 0, mii)
    WindowProc = 0
```

Case **WM_SYSCOMMAND**

' If Always On Top (ID = 1) was selected, change the on top/not on 
' top setting of Form1 to match.

```
    If wParam = 1 Then
        ' Reverse the setting and make it the current one.
        ontop = Not ontop
        retval = SetWindowPos(hwnd, IIf(ontop, HWND_TOPMOST, 
            HWND_NOTOPMOST), _
            0, 0, 0, 0, SWP_NOMOVE Or SWP_NOSIZE)
    WindowProc = 0
    Else
        ' Some other item was selected. Let the previous window 
        procedure 
        ' process it.
        WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, 
            lParam)
    End If
```

Case Else

' If this is some other message, let the previous procedure handle 
' it.

```
    WindowProc = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
End Select

End Function

' *** Place the following code inside Form1. ***

' When Form1 loads, add Always On Top to the system menu and set up the 
' new window procedure.

Private Sub Form_Load()
    Dim hSysMenu As Long     ' handle to the system menu
    Dim count As Long        ' the number of items initially on the menu
    Dim mii As MENUITEMINFO  ' describes a menu item to add
    Dim retval As Long       ' return value

    ' Get a handle to the system menu.
    hSysMenu = GetSystemMenu(Form1.hWnd, 0)
    ' See how many items are currently in it.
```
count = GetMenuItemCount(hSysMenu)

' Add a separator bar and then Always On Top to the system menu.
With mii
    ' The size of the structure.
    .cbSize = Len(mii)
    ' What parts of the structure to use.
    .fMask = MIIM_ID Or MIIM_TYPE
    ' This is a separator.
    .fType = MFT_SEPARATOR
    ' It has an ID of 0.
    .wID = 0
End With
' Add the separator to the end of the system menu.
retval = InsertMenuItem(hSysMenu, count, 1, mii)

' Likewise, add the Always On Top command.
With mii
    .fMask = MIIM_STATE Or MIIM_ID Or MIIM_TYPE
    ' This is a regular text item.
    .fType = MFT_STRING
    ' The option is enabled.
    .fState = MFS_ENABLED
    ' It has an ID of 1 (this identifies it in the window procedure).
    .wID = 1
    ' The text to place in the menu item.
    .dwTypeData = "&Always On Top"
    .cch = Len(.dwTypeData)
End With
' Add this to the bottom of the system menu.
retval = InsertMenuItem(hSysMenu, count + 1, 1, mii)

' Set the custom window procedure to process Form1's messages.
ontop = False
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
End Sub

' Before unloading, restore the default system menu and remove the
' custom window procedure.
Private Sub Form_Unload(Cancel As Integer)
  Dim retval As Long  ' return value
  
  ' Replace the previous window procedure to prevent crashing.
  retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
  ' Remove the modifications made to the system menu.
  retval = GetSystemMenu(Form1.hWnd, 1)
End Sub

See Also

WM_COMMAND
WWW_LBUTTONDBLCLK Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

The **WM_LBUTTONDBLCLK** message tells a window that the left mouse button has been double-clicked while the cursor is inside the window's **client area**. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the **GET_X_LPARAM**, **GET_Y_LPARAM**, and **MAKEPOINTS** macros can be used to unpack the coordinate information easily.

Return Value

**WM_LBUTTONDBLCLK** should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the **WM_LBUTTONDBLCLK** message. The DblClick event handler provided by Visual Basic actually process **WM_LBUTTONDBLCLK** and unpacks some of the information passed with it for easier use.

Parameters

**wParam**

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- **MK_CONTROL**: The Ctrl key is down.
- **MK_LBUTTON**: The left mouse button is down.
- **MK_MBUTTON**: The middle mouse button is down.
- **MK_RBUTTON**: The right mouse button is down.
Windows API Guide: WM_LBUTTONDOWN Message

The right mouse button is down.

MK_SHIFT
The Shift key is down.

MK_XBUTTON1
  Windows 2000: The first X button is down.

MK_XBUTTON2
  Windows 2000: The second X button is down.

IParam
The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_LBUTTONDOWN = &H203
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the left mouse button has been
double-clicked in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long ' x and y coordinates of the faked cursor position
Dim packed As Long ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT ' receives coordinates of the window
Dim retval As Long ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the right mouse button was just double-clicked in that position.
retval = SendMessage(Form1.hWnd, WM_LBUTTONDOWN, ByVal CLng(MK_LBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_LBUTTONUP, ByVal CLng(0), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_LBUTTONDOWN, ByVal CLng(MK_LBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_LBUTTONDOWN, ByVal CLng(0), ByVal packed)

See Also

WM_LBUTTONDOWN, WM_LBUTTONUP, WM_RBUTTONDOWN, WM_RBUTTONUP, WM_MBUTTONDOWN, WM_MBUTTONUP

Category

Mouse

Back to the Message list.
Back to the Reference section.
WM_LBUTTONDOWN Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The WM_LBUTTONDOWN message tells a window that the left mouse button has been pressed while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the GET_X_LPARAM, GET_Y_LPARAM, and MAKEPOINTS macros can be used to unpack the coordinate information easily.

Return Value

WM_LBUTTONDOWN should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the WM_LBUTTONDOWN message. The MouseDown event handler provided by Visual Basic actually process WM_LBUTTONDOWN and unpacks some of the information passed with it for easier use.

Parameters

wParam

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- MK_CONTROL
  The Ctrl key is down.
- MK_LBUTTON
  The left mouse button is down.
MK_MBUTTON
    The middle mouse button is down.
MK_RBUTTON
    The right mouse button is down.
MK_SHIFT
    The Shift key is down.
MK_XBUTTON1
    **Windows 2000:** The first X button is down.
MK_XBUTTON2
    **Windows 2000:** The second X button is down.

*Param*

The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

**Constant Definitions**

Const WM_LBUTTONDOWN = &H201
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

**Example**

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/w/wm_lbuttondown.html).

' Make window Form1 think that the left mouse button has been pressed in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long ' x and y coordinates of the faked cursor position
Dim packed As Long ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT ' receives coordinates of the window
Dim retval As Long ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the left mouse button was just pressed in that position.
retval = SendMessage(Form1.hWnd, WM_LBUTTONDOWN, ByVal CLng(MK_LBUTTON), ByVal packed)

See Also

WM_LBUTTONDOWNDBLCLK, WM_LBUTTONDOWNUP, WM_MBUTTONDOWN, WM_RBUTTONDOWN

Category

Mouse

Back to the Message list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_lbuttondown.html
WM_LBUTTONDOWN Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The WM_LBUTTONDOWN message tells a window that the left mouse button has been released while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the GET_X_LPARAM, GET_Y_LPARAM, and MAKEPOINTS macros can be used to unpack the coordinate information easily.

Return Value

WM_LBUTTONDOWN should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the WM_LBUTTONDOWN message. The MouseUp event handler provided by Visual Basic actually process WM_LBUTTONDOWN and unpacks some of the information passed with it for easier use.

Parameters

wParam

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- MK_CONTROL
  - The Ctrl key is down.
- MK_MBUTTON
The middle mouse button is down.
MK_RBUTTON
The right mouse button is down.
MK_SHIFT
The Shift key is down.
MK_XBUTTON1
Windows 2000: The first X button is down.
MK_XBUTTON2
Windows 2000: The second X button is down.

lParam
The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_LBUTTONDOWN = &H202
Const MK_CONTROL = &H8
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the left mouse button has been released in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long  ' x and y coordinates of the faked cursor position
Dim packed As Long  ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT  ' receives coordinates of the window
Dim retval As Long  ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the left mouse button was just released in that position.
retval = SendMessage(Form1.hWnd, WM_LBUTTONDOWN, ByVal CLng(0), ByVal packed)
See Also

WM_LBUTTONDOWN, WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN

Category

Mouse

Back to the Message list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_lbuttonup.html
WM_MBUTTONDBLCLK Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The `WM_MBUTTONDBLCLK` message tells a window that the middle mouse button has been double-clicked while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the `GET_X_LPARAM`, `GET_Y_LPARAM`, and `MAKEPOINTS` macros can be used to unpack the coordinate information easily.

Return Value

`WM_MBUTTONDBLCLK` should always return 0.

Visual Basic-Specific Issues

None.

Parameters

`wParam`

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- **MK_CONTROL**
  The Ctrl key is down.
- **MK_LBUTTON**
  The left mouse button is down.
- **MK_MBUTTON**
  The middle mouse button is down.
- **MK_RBUTTON**
  The right mouse button is down.
- **MK_SHIFT**
  The Shift key is down.
MK_XBUTTON1

Windows 2000: The first X button is down.
MK_XBUTTON2

Windows 2000: The second X button is down.

LPARAM
The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_MBUTTONDOWN = &H209
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the middle mouse button has been double-clicked in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long  ' x and y coordinates of the faked cursor position
Dim packed As Long  ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT  ' receives coordinates of the window
Dim retval As Long  ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the middle mouse button was just double-clicked in that position.
retval = SendMessage(Form1.hWnd, WM_MBUTTONDOWN, ByVal CLng(MK_MBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_MBUTTONUP, ByVal CLng(0), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_MBUTTONDOWN, ByVal CLng(MK_MBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_MBUTTONUP, ByVal CLng(0), ByVal packed)

See Also
Category

Mouse

Back to the Message list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_mbuttondblclk.html
WM_MBUTTONDOWN Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

The WM_MBUTTONDOWN message tells a window that the middle mouse button has been pressed while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the GET_X_LPARAM, GET_Y_LPARAM, and MAKEPOINTS macros can be used to unpack the coordinate information easily.

Return Value

WM_MBUTTONDOWN should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the WM_MBUTTONDOWN message. The MouseDown event handler provided by Visual Basic actually process WM_MBUTTONDOWN and unpacks some of the information passed with it for easier use.

Parameters

wParam

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- MK_CONTROL
  - The Ctrl key is down.
- MK_LBUTTON
  - The left mouse button is down.
MK_MBUTTON
   The middle mouse button is down.
MK_RBUTTON
   The right mouse button is down.
MK_SHIFT
   The Shift key is down.
MK_XBUTTON1
   Windows 2000: The first X button is down.
MK_XBUTTON2
   Windows 2000: The second X button is down.

LPARAM
   The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_MBUTTONDOWN = &H207
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the middle mouse button has been pressed in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long ' x and y coordinates of the faked cursor position
Dim packed As Long ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT ' receives coordinates of the window
Dim retval As Long ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value packed = (ycoord * &H10000) + xcoord

' Make Form1 think the middle mouse button was just pressed in that position.
retval = SendMessage(Form1.hWnd, WM_MBUTTONDOWN, ByVal CLng(MK_MBUTTON), ByVal packed)

See Also

WM_LBUTTONDOWN, WM_MBUTTONDOWNBLCLK, WM_MBUTTONUP, WM_RBUTTONDOWN

Category

Mouse

Back to the Message list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_mbuttondown.html
WM_MBUTTONDOWN Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

The **WM_MBUTTONDOWN** message tells a window that the middle mouse button has been released while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the **GET_X_LPARAM**, **GET_Y_LPARAM**, and **MAKEPOINTS** macros can be used to unpack the coordinate information easily.

Return Value

**WM_MBUTTONDOWN** should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the **WM_MBUTTONDOWN** message. The MouseUp event handler provided by Visual Basic actually process **WM_MBUTTONDOWN** and unpacks some of the information passed with it for easier use.

Parameters

**wParam**

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- **MK_CONTROL**
  The Ctrl key is down.
- **MK_LBUTTON**
  The left mouse button is down.
MK_RBUTTON
  The right mouse button is down.
MK_SHIFT
  The Shift key is down.
MK_XBUTTON1
  Windows 2000: The first X button is down.
MK_XBUTTON2
  Windows 2000: The second X button is down.

IParam
  The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_MBUTTONDOWN = &H208
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the middle mouse button has been released in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long ' x and y coordinates of the faked cursor position
Dim packed As Long ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT ' receives coordinates of the window
Dim retval As Long ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the middle mouse button was just released in that position.
retval = SendMessage(Form1.hWnd, WM_MBUTTONDOWN, ByVal CLng(0), ByVal packed)
See Also

WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN

Category

Mouse

Back to the Message list.
Back to the Reference section.
WM_MOUSEMOVE Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The WM_MOUSEMOVE message tells a window that the position of the mouse cursor inside of its boundaries has moved. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the GET_X_LPARAM, GET_Y_LPARAM, and MAKEPOINTS macros can be used to unpack the coordinate information easily.

Return Value

WM_MOUSEMOVE should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special hander for the WM_MOUSEMOVE message. The MouseMove event handler provided by Visual Basic actually process WM_MOUSEMOVE and unpacks some of the information passed with it for easier use.

Parameters

\( wParam \)

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- MK_CONTROL
  - The Ctrl key is down.
- MK_LBUTTON
The left mouse button is down.
MK_MBUTTON
   The middle mouse button is down.
MK_RBUTTON
   The right mouse button is down.
MK_SHIFT
   The Shift key is down.
MK_XBUTTON1
   Windows 2000: The first X button is down.
MK_XBUTTON2
   Windows 2000: The second X button is down.

IParam
   The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_MOUSEMOVE = &H200
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the mouse cursor has moved into its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long   ' x and y coordinates of the faked cursor position
Dim packed As Long   ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT   ' receives coordinates of the window
Dim retval As Long   ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord
' Make Form1 think the cursor just moved to that position.
retval = SendMessage(Form1.hWnd, WM_MOUSEMOVE, ByVal CLng(0), ByVal packed)

Category

Mouse

Back to the Message list.
Back to the Reference section.
WM_RBUTTONDOWNDBLCLK Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The **WM_RBUTTONDOWNDBLCLK** message tells a window that the right mouse button has been double-clicked while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the **GET_X_LPARAM**, **GET_Y_LPARAM**, and **MAKEPOINTS** macros can be used to unpack the coordinate information easily.

Return Value

**WM_RBUTTONDOWNDBLCLK** should always return 0.

Visual Basic-Specific Issues

None.

Parameters

**wParam**

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- **MK_CONTROL**
  - The Ctrl key is down.
- **MK_LBUTTON**
  - The left mouse button is down.
- **MK_MBUTTON**
  - The middle mouse button is down.
- **MK_RBUTTON**
  - The right mouse button is down.
- **MK_SHIFT**
The Shift key is down.

**Windows X**

MK_XBUTTON1

Windows 2000: The first X button is down.
MK_XBUTTON2

Windows 2000: The second X button is down.

**lParam**

The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

### Constant Definitions

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>WM_RBUTTONDBLCLK</td>
<td>&amp;H206</td>
</tr>
<tr>
<td>MK_CONTROL</td>
<td>&amp;H8</td>
</tr>
<tr>
<td>MK_LBUTTON</td>
<td>&amp;H1</td>
</tr>
<tr>
<td>MK_MBUTTON</td>
<td>&amp;H10</td>
</tr>
<tr>
<td>MK_RBUTTON</td>
<td>&amp;H2</td>
</tr>
<tr>
<td>MK_SHIFT</td>
<td>&amp;H4</td>
</tr>
<tr>
<td>MK_XBUTTON1</td>
<td>&amp;H20</td>
</tr>
<tr>
<td>MK_XBUTTON2</td>
<td>&amp;H40</td>
</tr>
</tbody>
</table>

### Example

```vbs
' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the right mouse button has been
double-clicked in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long ' x and y coordinates of the faked cursor position
Dim packed As Long ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT ' receives coordinates of the window
Dim retval As Long ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the right mouse button was just double-clicked in that position.
retval = SendMessage(Form1.hWnd, WM_RBUTTONDOWN, ByVal CLng(MK_RBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_RBUTTONUP, ByVal CLng(0), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_RBUTTONDBLCLK, ByVal CLng(MK_RBUTTON), ByVal packed)
retval = SendMessage(Form1.hWnd, WM_RBUTTONUP, ByVal CLng(0), ByVal packed)
```

See Also

WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN, WM_RBUTTONUP

Category

Mouse

Back to the Message list.
Back to the Reference section.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_rbuttondblclk.html
WM_RBUTTONDOWN Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The **WM_RBUTTONDOWN** message tells a window that the right mouse button has been pressed while the cursor is inside the window's client area. The information sent with the message identifies the cursor position relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the **GET_X_LPARAM**, **GET_Y_LPARAM**, and **MAKEPOINTS** macros can be used to unpack the coordinate information easily.

Return Value

**WM_RBUTTONDOWN** should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special handler for the **WM_RBUTTONDOWN** message. The MouseDown event handler provided by Visual Basic actually process **WM_RBUTTONDOWN** and unpacks some of the information passed with it for easier use.

Parameters

**wParam**

A combination of the following flags specifying which modifier keys, if any, are currently depressed:

- **MK_CONTROL**
  - The Ctrl key is down.
- **MK_LBUTTON**
  - The left mouse button is down.
MK_MBUTTON
   The middle mouse button is down.
MKRBUTTON
   The right mouse button is down.
MK_SHIFT
   The Shift key is down.
MKXBUTTON1
   Windows 2000: The first X button is down.
MKXBUTTON2
   Windows 2000: The second X button is down.

lParam
   The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

Constant Definitions

Const WM_RBUTTONDOWN = &H204
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_RBUTTON = &H2
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40

Example

' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the right mouse button has been pressed in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long   ' x and y coordinates of the faked cursor position
Dim packed As Long   ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT   ' receives coordinates of the window
Dim retval As Long   ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect(Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the right mouse button was just pressed in that position.
retval = `SendMessage(Form1.hWnd, WM_RBUTTONDOWN, ByVal CLng(MK_RBUTTON), ByVal packed)`

See Also

WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDBLCLK, WM_RBUTTONUP

Category

Mouse

Back to the Message list.
Back to the Reference section.
WM_RBUTTONUP Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Not Supported.

Description & Usage

The WM_RBUTTONUP message tells a window that the right mouse button has been released while the cursor is inside the window's client area. The information sent with the message identifies the cursor postion relative to the window as well as the state of various modifier keys and mouse buttons. The target window's window procedure processes the message. When handling the message, the GET_X_LPARAM, GET_Y_LPARAM, and MAKEPOINTS macros can be used to unpack the coordinate information easily.

Return Value

WM_RBUTTONUP should always return 0.

Visual Basic-Specific Issues

It is not necessary to create a special hander for the WM_RBUTTONUP message. The MouseUp event handler provided by Visual Basic actually process WM_RBUTTONUP and unpacks some of the information passed with it for easier use.

Parameters

wParam

A combination of the following flags specifying which modifier keys, if any, are currently depressed:
- MK_CONTROL
  - The Ctrl key is down.
- MK_LBUTTON
The left mouse button is down.
MK_MBUTTON
  The middle mouse button is down.
MK_SHIFT
  The Shift key is down.
MK_XBUTTON1
  Windows 2000: The first X button is down.
MK_XBUTTON2
  Windows 2000: The second X button is down.

`lParam`
The (x,y) coordinates of the mouse cursor relative to the window. The low-order word contains the x-coordinate, and the high-order word contains the y-coordinate.

**Constant Definitions**

```vba
Const WM_RBUTTONUP = &H205
Const MK_CONTROL = &H8
Const MK_LBUTTON = &H1
Const MK_MBUTTON = &H10
Const MK_SHIFT = &H4
Const MK_XBUTTON1 = &H20
Const MK_XBUTTON2 = &H40
```

**Example**

```vba
' This code is licensed according to the terms and conditions listed here.

' Make window Form1 think that the right mouse button has been released in its center by sending the appropriate message to it.
Dim xcoord As Long, ycoord As Long  ' x and y coordinates of the faked cursor position
Dim packed As Long  ' the coordinates "packed" into a single 32-bit integer
Dim winrect As RECT  ' receives coordinates of the window
Dim retval As Long  ' return value

' First, get the coordinates of window Form1.
retval = GetWindowRect (Form1.hWnd, winrect)
' Use the coordinates to calculate the midpoint of Form1.
xcoord = (winrect.right - winrect.left) / 2
ycoord = (winrect.bottom - winrect.top) / 2
' Now pack the coordinates into the appropriate words of the value
packed = (ycoord * &H10000) + xcoord

' Make Form1 think the right mouse button was just released in that position.
retval = SendMessage (Form1.hWnd, WM_RBUTTONUP, ByVal CLng(0), ByVal packed)
```
See Also

WM_LBUTTONDOWN, WM_MBUTTONDOWN, WM_RBUTTONDOWN, WM_RBUTTONDBLCLK, WM_RBUTTONUP, WM_RBUTTONDOWN

Category

Mouse

Back to the Message list.
Back to the Reference section.
WM_Timer Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

A window receives the WM_TIMER message when the time-out period of a timer (created by the SetTimer function) elapses. You should only create a handler for this message if you did not specify a TimerProc callback function for the timer. Otherwise, allow the window's default message handler to invoke the TimerProc function.

Return Value

The WM_TIMER message should always return 0.

Visual Basic-Specific Issues

None.

Parameters

wParam
  The unique identifier of the timer that sent the message.

lParam
  A pointer to the TimerProc callback function that should be invoked. If no function is specified, this will be 0.

Constant Definitions

Const WM_TIMER = &H113

Example

Display the current time in text box control Text1. The time is updated twice every second, and the time is formatted according
to the current locale's settings. To use this example, place a text edit box named Text1 on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Type SYSTEMTIME
    wYear As Integer
    wMonth As Integer
    wDayOfWeek As Integer
    wDay As Integer
    wHour As Integer
    wMinute As Integer
    wSecond As Integer
    wMilliseconds As Integer
End Type
Public Declare Function SetTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long, ByVal uElapse As Long, ByVal lpTimerFunc As Long) As Long
Public Declare Function KillTimer Lib "user32.dll" (ByVal hWnd As Long, ByVal nIDEvent As Long) As Long
Public Const WM_TIMER = &H113
Public Declare Function GetTimeFormat Lib "kernel32.dll" Alias "GetTimeFormatA" (ByVal Locale As Long, ByVal dwFlags As Long, ByVal lpTime As SYSTEMTIME, ByVal lpFormat As Any, ByVal lpTimeStr As String, ByVal cchTime As Long) As Long
Public Declare Function SetWindowLong Lib "user32.dll" Alias "SetWindowLongA" (ByVal hWnd As Long, ByVal nIndex As Long, ByVal dwNewLong As Long) As Long
Public Const GWL_WNDPROC = -4
Public Declare Function CallWindowProc Lib "user32.dll" Alias "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hWnd As Long, ByVal Msg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long

' *** Place the following code inside a module. ***

' A pointer to Form1's default window procedure:
Public pOldProc As Long

' The following function will handle all of Form1's messages. It only
' actually processes WM_TIMER and passes all others to the default handler.
Public Function WindowProc (ByVal hWnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim systime As SYSTEMTIME  ' the current time
    Dim timestr As String * 260  ' receives the formatted string
    Dim slength As Long  ' length of formatted string returned
    Select Case uMsg
Case WM_TIMER
  ' Retrieve the current time, according to the computer's time zone.
  GetLocalTime systime
  ' Format a string to represent the time.
  slength = GetTimeFormat(0, 0, systime, CLng(0), timestr, Len(timestr))
  ' Display the string in Text1, found on window Form1.
  Form1.Text1.Text = Left(timestr, slength)
  WindowProc = 0
Case Else
  ' Pass the message to Form1's previous procedure.
  WindowProc = CallWindowProc(pOldProc, hWnd, uMsg, wParam, lParam)
End Select
End Function

' *** Place the following code inside Form1. ***

' Create the timer when the form opens and destroy it when the form closes.
' The timer is given an ID of 1, so the return values don't need to be saved.
' Also, set up the window procedure when the form is loaded, and remove it
' when the form is unloaded. This procedure will receive all of Form1's messages.
Private Sub Form1_Load()
  Dim retval As Long  ' return value
  pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)
  retval = SetTimer(Form1.hWnd, 1, 500, AddressOf TimerProc)
End Sub

Private Sub Form1_Unload(Cancel As Integer)
  Dim retval As Long  ' return value
  retval = KillTimer(Form1.hWnd, 1)
  retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)
End Sub

See Also

SetTimer

Category

Timers

Back to the Message list.
Back to the Reference section.

Last Modified: December 17, 2000
WM_CLOSE Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the WM_CLOSE message to a window, naturally enough, instructs it to close. Some windows will prompt the user with a confirmation dialog box before it closes, while others will close immediately. Exactly what happens depends on the program. Just keep in mind that sending WM_CLOSE merely asks the window to close -- there's no guarantee that will actually happen.

Return Value

This message should always return zero.

Visual Basic-Specific Issues

None.

Parameters

- **wParam**
  - Not used -- set to zero.

- **lParam**
  - Not used -- set to zero.

Constant Definitions

- **Const WM_CLOSE = &H10**

Example

Let the user enter the title of a window in a text box. Then, attempt to close the window whose title bar matches that text when
the user clicks a button. To use this example, place a text boxed named txtTitle and a command button named cmdClose on a form window.

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function SendMessage Lib "user32.dll" Alias "SendMessageA" (ByVal hWnd As Long, ByVal _
    Msg As Long, wParam As Any, lParam As Any) As Long
Public Const WM_CLOSE = &H10
Public Declare Function FindWindow Lib "user32.dll" Alias "FindWindowA" (ByVal lpClassName As String, _
    ByVal lpWindowName As String) As Long

' *** Place the following code inside the form window. ***

Private Sub cmdClose_Click ()
    Dim hWnd As Long  ' handle of the window to try to close
    Dim result As Long  ' result of the message

    ' Get a handle to the window whose title the user entered.
    hWnd = FindWindow(vbNullString, txtTitle.Text)
    If hWnd <> 0 Then
        ' Ask the window to close.
        result = SendMessage(hWnd, WM_CLOSE, ByVal CLng(0), ByVal CLng(0))
        Debug.Print "Told " & txtTitle.Text & " to close."
    Else
        ' Couldn't find the window the user specified.
        Debug.Print "Unable to find window titled " & txtTitle.Text & "."
    End If
End Sub

See Also

DestroyWindow

Category

Back to the Function list.
Back to the Reference section.

Last Modified: December 17, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_close.html
WM_GETTEXT Message

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Requires Windows CE 1.0 or later.

Description & Usage

Send the `WM_GETTEXT` message to a window to retrieve that window's text. For regular windows, this text is the text displayed in the title bar caption area. The target window's text is copied to a text buffer supplied in the message parameters. The message is handled by the target window's default window procedure.

Return Value

The message returns the number of characters retrieved, not counting the terminating null character.

Visual Basic-Specific Issues

When using `SendMessage` to send the `WM_GETTEXT` message, the ByVal keyword must be used in front of both the `wParam` and `lParam` parameters.

Parameters

`wParam`

The number of characters to copy from the target window's text, including the necessary terminating null character.

`lParam`

The string which receives the first `wParam` characters (minus one for the null) of the target window's text.

Constant Definitions

```
Const WM_GETTEXT = &H
```

Example
This code is licensed according to the terms and conditions listed here.

Display the title bar text of window Form1 by sending the appropriate messages to it.
Dim wintext As String ' receives the copied text from the target window
Dim slength As Long ' length of the window text
Dim retval As Long ' return value of message

' First, determine how much space is necessary for the buffer.
' (1 is added for the terminating null character.)
slength = SendMessage(Form1.hWnd, WM_GETTEXTLENGTH, ByVal CLng(0), ByVal CLng(0)) + 1
' Make enough room in the buffer to receive the text.
wintext = Space(slength)
' Copy the target window's text into the buffer.
retval = SendMessage(Form1.hWnd, WM_GETTEXT, ByVal slength, ByVal wintext)
' Remove the terminating null and extra space from the buffer.
wintext = Left(wintext, retval)
' Display the result.
Debug.Print "Form1's title bar text is: "; wintext

See Also

GetWindowText, WM_GETTEXTLENGTH, WM_SETTEXT

Category

Windows

Last Modified: January 26, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_gettext.html
WM_GETTEXTLENGTH Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Send the WM_GETTEXTLENGTH message to a window to discover the number of characters in that window's text. This character count does not include the terminating null character. In some cases, this message will report a larger number of characters than what actually exists; however, the count obtained by this message will always be no less than the actual number of characters. The WM_GETTEXTLENGTH message is handled by the target window's default window procedure.

Return Value

The message returns the number of characters in the target window's text, not including the terminating null.

Visual Basic-Specific Issues

When using SendMessage to send the WM_GETTEXTLENGTH message, both the wParam and lParam parameters must be set using the expression ByVal CLng(0).

Parameters

- wParam
  Not used -- set to 0.
- lParam
  Not used -- set to 0.

Constant Definitions

Const WM_GETTEXTLENGTH = &HE

Example
This code is licensed according to the terms and conditions listed here.

Display the title bar text of window Form1 by sending the appropriate messages to it.
Dim wintext As String ' receives the copied text from the target window
Dim slength As Long ' length of the window text
Dim retval As Long ' return value of message

' First, determine how much space is necessary for the buffer.
' (1 is added for the terminating null character.)
slength = SendMessage(Form1.hWnd, WM_GETTEXTLENGTH, ByVal CLng(0), ByVal CLng(0)) + 1

' Make enough room in the buffer to receive the text.
wintext = Space(slength)

' Copy the target window's text into the buffer.
retval = SendMessage(Form1.hWnd, WM_GETTEXT, ByVal slength, ByVal wintext)

' Remove the terminating null and extra space from the buffer.
wintext = Left(wintext, retval)

' Display the result.
Debug.Print "Form1's title bar text is: "; wintext

See Also

GetWindowTextLength, WM_GETTEXT

Category

Windows

Back to the Message list.
Back to the Reference section.

Last Modified: January 27, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_gettextlength.html
WM_HELP Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.51 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

The WM_HELP message instructs a window to display context-sensitive help. This help could refer to the window itself, one of the controls on it, or a child window or dialog it created. The message includes a structure identifying the object which generated the message. While a window does not necessarily have to respond to the WM_HELP message by opening a WinHelp document, it ought to display help information of some kind.

Return Value

The WM_HELP message should always return a non-zero value.

Visual Basic-Specific Issues

None.

Parameters

wParam
Not used -- set to 0.

lParam
A pointer to a HELPINFO structure identifying the control or other window which originated the message.

Constant Definitions

Const WM_HELP = &H53

Example
This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/w/wm_help.html).

Display an HTML document to provide help when the user clicks the "Help" button of a dialog box. Notice how the WM_HELP message must be handled explicitly in this example, since Visual Basic does not allow you to create a handler through the interface. Pay careful attention to where each piece of code must go.

*** Place the following code in a module. ***

This is a pointer to Form1's previous window procedure.

```vbnet
Public pOldProc As Long
```

This is the handler for the WM_HELP message.

```vbnet
Public Function WindowProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim hi As HELPINFO ' information about window requesting help
    Dim slength As Long ' length of string
    Dim wintext As String ' text of window calling for help
    Dim retval As Long ' return value

    ' Decide which message has been received.
    Select Case uMsg
        Case WM_HELP
            ' Determine if the message box, having text "Warning!", is requesting help. If so, display an HTML document for help. Note: in a "real" program, you should use the MessageBoxIndirect function because it allows a Context ID for a WinHelp file to be specified. But since this is an example for WM_HELP, this inferior method is presented.

            ' Copy the information about the help message into the structure.
            CopyMemory hi, ByVal lParam, Len(hi)

            ' Determine the text of the window for which help is requested.
            slength = GetWindowTextLength(hi.hItemHandle) + 1
            wintext = Space(slength)
            retval = GetWindowText(hi.hItemHandle, wintext, slength)
            wintext = Left(wintext, retval)

            ' If it is "Warning!", open up the proper HTML document.
            If wintext = "Warning!" Then
                retval = ShellExecute(hwnd, "open", "C:\MyProg\mboxhelp.html", ",", ",", SM_RESTORE)
            End If

            ' Return successfully.
            WindowProc = 1
        Case Else
    End Select
```

Let the previous message handler process this message.
retval = CallWindowProc(pOldProc, hwnd, uMsg, wParam, lParam)
WindowProc = retval
End Select
End Function

*** Place the following code in the Form1_Load procedure. ***
' Set our custom window procedure as Form1's procedure.
pOldProc = SetWindowLong(Form1.hWnd, GWL_WNDPROC, AddressOf WindowProc)

*** Place the following code in the Form1_Unload procedure. ***
' Set the previous procedure as the one Form1 uses (to make VB happy).
Dim retval As Long
retval = SetWindowLong(Form1.hWnd, GWL_WNDPROC, pOldProc)

*** Place the following code wherever you want to invoke the message box. ***
' Prompt the user for a selection, allowing him to get help about his choice.
Dim mbresult As Long ' result of message box
Dim flags As Long ' message box's flags

flags = MB_YESNO Or MB_HELP Or MB_ICONWARNING
mbresult = MessageBox(Form1.hWnd, "Are you sure?", "Warning!", flags)
If mbresult = IDYES Then
    Debug.Print "You said 'Yes'"
Else
    Debug.Print "You said 'No'"
End If

Category

Windows

Back to the Message list.
Back to the Reference section.

Last Modified: January 31, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/w/wm_help.html
WM_SETTEXT Message

Platforms

- Windows 95: Supported.
- Windows 98: Supported.
- Windows NT: Requires Windows NT 3.1 or later.
- Windows CE: Requires Windows CE 1.0 or later.

Description & Usage

Sending the WM_SETTEXT message to a window sets that window's text. For ordinary windows, the text is the text that appears in the title bar. The message is handled by the target window's default window procedure.

Return Value

If the message returns 0, an error occurred. If the message returns a non-zero value, the window text was successfully set.

Visual Basic-Specific Issues

When sending the WM_SETTEXT message, the expression ByVal CLng(0) must be used for the wParam parameter. Additionally, the ByVal keyword must also precede the string passed as the lParam parameter.

Parameters

wParam
- Not used -- set to 0.

lParam
- The string to set as the window text.

Constant Definitions
Const WM_SETTEXT = &HC

Example

' This code is licensed according to the terms and conditions listed here.

' Set the title bar text of window Form1 to "Caption Area".
' Do this by sending the proper message to a window.
Dim wintext As String  ' string to set as the window's text
Dim retval As Long  ' return value of message

' Store the desired text in a string.
wintext = "Caption Area"
' Send a message telling window Form1 to make that string its text.
retval = SendMessage(Form1.hWnd, WM_SETTEXT, ByVal CLng(0), ByVal wintext)

' Display the result of the message.
If retval = 0 Then
    Debug.Print "Unable to set the text of Form1."
Else
    Debug.Print "The text of Form1 was successfully set."
End If

See Also

SetWindowText, WM_GETTEXT

Category

Windows

Back to the Message list.
Back to the Reference section.

Last Modified: January 25, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
Windows API Reference: Messages

Last Update: January 21, 2001
Number of Messages Listed: 66 (6 added)

Below is an alphabetical list of the API messages currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to: A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

- **B**
  - BM_CLICK
  - BM_GETCHECK
  - BM_GETSTATE
  - BM_SETCHECK
  - BM_SETSTATE

- **C**
  - CB_ADDSTRING
  - CB_DELETESTRING
  - CB_GETCOUNT
  - CB_GETCURSEL
  - CB_GETDROPPEDSTATE
  - CB_GETLBTEXT
  - CB_GETLBTEXTLEN
  - CB_INSERTSTRING
  - CB_RESETCONTENT
  - CB_SETCURSEL
  - CB_SHOWDROPDOWN
EM_CANUNDO
EM_GETFIRSTVISIBLELINE
EM_GETLINE
EM_GETPASSWORDCHAR
EM_GETSEL
EM_LINEINDEX
EM_LINELENGTH
EM_REPLACESEL
EM_SETPASSWORDCHAR
EM_SETSEL
EM_UNDO
IPM_CLEARADDRESS
IPM_GETADDRESS
IPM_ISBLANK
IPM_SETADDRESS
IPM_SETFOCUS
IPM_SETFOCUS
LB_ADDSTRING
LB_DELETESTRING
LB_GETCOUNT
LB_GETCURSEL NEW
LB_GETSEL NEW
LB_GETSELNEW
LB_GETSELCOUNT NEW
LB_GETSELITEMS NEW
LB_GETTEXT
LB_GETTEXTLEN
LB_INSERTSTRING
LB_RESETCONTENT
LB_SETCURSEL NEW
LB_SETSEL NEW
MM_MCINOTIFY
WM_CLOSE
WM_COMMAND
WM_GETTEXT
WM_GETTEXTLENGTH
WM_HELP
WM_INITMENU
WM_LBUTTONDOWN
WM_LBUTTONUP
WM_MBUTTONDOWN
WM_MBUTTONUP
WM_MOUSEMOVE
WM_RBUTTONDOWN
WM_RBUTTONUP
WM_SETTEXT
WM_SYSCOMMAND
WM_TIMER

Go back to the Reference section index.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/msga.html
Below is an alphabetical list of the API callback functions currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to: A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

- B
  - BrowseCallbackProc
- C
  - CCHookProc
  - CFHookProc
- E
  - EnumChildProc
  - EnumFontFamExProc
  - EnumFontFamProc
  - EnumThreadWndProc
  - EnumWindowsProc
- M
  - MsgBoxCallback
- O
  - OFNHookProc
  - OFNHookProcOldStyle
- P
- **PropEnumProcEx**
- **S**
  - **SoundSentryProc**
- **T**
  - **TimerProc**
- **W**
  - **WindowProc**

Go back to the Reference section index.

---

**Last Modified:** January 21, 2001

This page is copyright © 2001 Paul Kuliniewicz. [Copyright Information](mailto:vbapi@vbapi.com) [Revised October 29, 2000](http://www.vbapi.com/ref/calla.html) [Go back to the Windows API Guide home page.](http://216.26.168.92/vbapi/ref/calla.html)

E-mail: vbapi@vbapi.com [Send Encrypted E-Mail](mailto:vbapi@vbapi.com)

This page is at [http://www.vbapi.com/ref/calla.html](http://www.vbapi.com/ref/calla.html)
BrowseCallbackProc Callback Function

Public Function BrowseCallbackProc(ByVal hwnd As Long, ByVal uMsg As Long, ByVal lParam As Long, ByVal lpData As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

The **BrowseCallbackProc** callback function process the messages of a Browse for Folder dialog box. Specifically, this function responds to the messages notifying when the dialog box is being initialized and when the user as changed the current selection. This function does not necessarily have to be called **BrowseCallbackProc**; that is merely the name given to it in discussions about the API. The callback function can work with the dialog box by sending it one of the following messages via **SendMessage**:

**BFFM_ENABLEOK**
- Enable or disable the dialog box's OK button. To enable the OK button, set the **lParam** message parameter to a non-zero value. To disable the OK button, set the **lParam** message parameter to 0.

**BFFM_SETSELECTION**
- Set the current selection in the dialog box. To specify the desired path using a string, set the **lParam** message parameter to the string and the **wParam** message parameter to a non-zero value. To specify the desired path using a pointer to an **ITEMIDLIST** structure (a.k.a. a PIDL), set the **lParam** message parameter to the PIDL and the **wParam** message parameter to 0.

**BFFM_SETSTATUSTEXT**
- Set the status text displayed by the dialog box, if it exists. Set the **lParam** message parameter to the string holding the desired text.

Return Value

The function always returns 0, unless the dialog box is processing the **BFFM_VALIDATEFAILED** message (see that message's description below for details).

Visual Basic-Specific Issues

Like all callback functions, **BrowseCallbackProc** must be declared Public within a module.

Parameters
hwnd

A **handle** to the Browse for Folder dialog box calling this function. This handle is necessary to send messages back to the dialog box.

uMsg

One of the following flags specifying the event:

- **BFFM_INITIALIZED**
  The dialog box has just finished initializing. *lParam* is 0.
- **BFFM_SELCHANGED**
  The user has changed the current selection. *lParam* is a PIDL to the current selection.
- **BFFM_VALIDATEFAILED**
  With Internet Explorer 4.0 or later: The user has typed an invalid path into the edit box. *lParam* is a pointer to a null-terminated string holding the invalid path name. The function should return 0 to close the dialog box, or a non-zero value to keep it displayed.

lParam

Depends on the value of *uMsg*.

lpData

The application-defined value given in the **BROWSEINFO** structure used to create the dialog box.

## Constant Definitions

```
Const BFFM_ENABLEOK = &H465
Const BFFM_SETSELECTION = &H466
Const BFFM_SETSTATUSTEXT = &H464
Const BFFM_INITIALIZED = 1
Const BFFM_SELCHANGED = 2
Const BFFM_VALIDATEFAILED = 3
```

## Used By

**BROWSEINFO**

Go back to the Callback Function listing.
Go back to the Reference section index.

**Last Modified:** September 24, 1999

This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information](http://www.vbapi.com/ref/b/browsecallbackproc.html) Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at [http://www.vbapi.com/ref/b/browsecallbackproc.html](http://www.vbapi.com/ref/b/browsecallbackproc.html)
CCHookProc Callback Function

Public Function CCHookProc(ByVal hdlg As Long, ByVal uiMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

CCHookProc is used to process a Choose Color common dialog box's messages. This hook function allows a program to write its own code to process messages otherwise handled by the Choose Color box. The function does not have to be named CCHookProc -- that is merely the name given to it in discussions about the API.

Return Value

If the function returns 0, the Choose Color dialog box's default message handler processes the message. If the function returns a non-zero value, the Choose Color dialog box's default message handler does not receive the message.

Visual Basic-Specific Issues

Like all callback functions, CCHookProc must be declared Public and be defined in a module.

Parameters

hdlg
    A handle to the Choose Color dialog box which the message is for.
uiMsg
    The message to be acted upon.
wParam
    Additional information about the message.
lParam
    Additional information about the message.

Used By

CHOOSECOLOR_TYPE

Go back to the Callback Function listing.
CFHookProc Callback Function

Public Function CFHookProc(ByVal hdlg As Long, ByVal uiMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
' application-defined code goes here
End Function

Description & Usage

CFHookProc is used to process a Choose Font common dialog box's messages. This hook function allows a program to write its own code to process messages otherwise handled by the Choose Font box. The function does not have to be named CFHookProc -- that is merely the name given to it in discussions about the API.

Return Value

If the function returns 0, the Choose Font dialog box's default message handler processes the message. If the function returns a non-zero value, the Choose Font dialog box's default message handler does not receive the message.

Visual Basic-Specific Issues

Like all callback functions, CFHookProc must be declared Public and be defined in a module.

Parameters

hdlg
A handle to the Choose Font dialog box which the message is for.

uiMsg
The message to be acted upon.

wParam
Additional information about the message.

lParam
Additional information about the message.

Used By

CHOOSEFONT_TYPE

Go back to the Callback Function listing.
EnumChildProc Callback Function

Public Function EnumChildProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

EnumChildProc is an application-defined callback function used by EnumChildWindows for each window enumerated. This function should process the window in whatever manner is required. This function does not necessarily have to be named EnumChildProc -- that is merely the name given to it in discussions about the API.

Return Value

If the function returns 0, EnumChildWindows will immediately stop enumerating child windows. If the function returns a non-zero value, EnumChildWindows will continue enumerating child windows until it can no longer find any.

Visual Basic-Specific Issues

Like all callback functions, EnumChildProc must be declared Public and be defined in a module.

Parameters

hwnd
A handle to the window which EnumChildWindows has just found.

lParam
Whatever value was specified in the parameter list of EnumChildWindows.

Used By

EnumChildWindows

Go back to the Callback Function listing.
Go back to the Reference section index.
EnumFontFamExProc Callback Function

Public Function EnumFontFamExProc (ByVal lpelfe As Long, ByVal lpntme As Long, ByVal FontType As Long, ByVal lParam As Long) As Long
    ' application-specific code goes here
End Function

Description & Usage

The EnumFontFamExProc callback function processes each font enumerated by EnumFontFamiliesEx. The function receives a large amount of information describing both the logical font attributes as well as its metrics. Information describing TrueType fonts is received by the function in a slightly different format than non-TrueType font information. This function does not necessarily have to be named EnumFontFamExProc -- that is simply the name given to it in discussions about the Windows API.

Return Value

If the function returns 0, EnumFontFamiliesEx immediately stops enumerating fonts. If the function returns a non-zero value, EnumFontFamiliesEx continues enumerating fonts as long as another one can be found.

Visual Basic-Specific Issues

Like all callback functions, EnumFontFamExProc must be declared Public and be defined in a module.

Parameters

lpelfe
A pointer to an ENUMLOGFONTEX structure describing the logical attributes of the font.

lpntme
If the font is a TrueType font, this is a pointer to a NEWTEXTMETRICEX structure. If the font is not a TrueType font, this is a pointer to a TEXTMETRIC structure. Either structure describes the text metrics of the font.

FontType
A combination of the following flags (although usually only one is used) identifying the type of font:

DEVICE_FONTTYPE
The font is a device font.

RASTER_FONTTYPE
The font is a raster font.

TRUETYPE_FONTTYPE
The font is a TrueType font.

lParam
Windows API Guide: EnumFontFamExProc Callback Function

Additional data specified by EnumFontFamiliesEx.

**Constant Definitions**

Const DEVICE_FONTTYPE = &H2
Const RASTER_FONTTYPE = &H1
Const TRUETYPE_FONTTYPE = &H4

**Used By**

EnumFontFamiliesEx

Go back to the Callback Function listing.
Go back to the Reference section index.
EnumFontFamProc Callback Function

Public Function EnumFontFamProc (ByVal lpelf As Long, ByVal lpntm As Long, ByVal FontType As Long, ByVal lParam As Long) As Long
' application-specific code goes here
End Function

Description & Usage

The EnumFontFamProc callback function processes each font enumerated by EnumFontFamilies. The function receives a large amount of information describing both the logical font attributes as well as its metrics. Information describing TrueType fonts is received by the function in a slightly different format than non-TrueType font information. This function does not necessarily have to be named EnumFontFamProc -- that is simply the name given to it in discussions about the Windows API.

Return Value

If the function returns 0, EnumFontFamilies immediately stops enumerating fonts. If the function returns a non-zero value, EnumFontFamilies continues enumerating fonts as long as another one can be found.

Visual Basic-Specific Issues

Like all callback functions, EnumFontFamProc must be declared Public and be defined in a module.

Parameters

lpelf
A pointer to an ENUMLOGFONT structure describing the logical attributes of the font.

lpntm
If the font is a TrueType font, this is a pointer to a NEWTEXTMETRIC structure. If the font is not a TrueType font, this is a pointer to a TEXTMETRIC structure. Either structure describes the text metrics of the font.

FontType
A combination of the following flags (although usually only one is used) identifying the type of font:

DEVICE_FONTTYPE
The font is a device font.

RASTER_FONTTYPE
The font is a raster font.
### Constant Definitions

```
Const DEVICE_FONTTYPE = &H2
Const RASTER_FONTTYPE = &H1
Const TRUETYPE_FONTTYPE = &H4
```

### Used By

- EnumFontFamilies

Go back to the Callback Function listing.
Go back to the Reference section index.

---

**Last Modified:** October 29, 1999
This page is copyright © 1999 Paul Kuliniewicz. [Copyright Information] Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com [Send Encrypted E-Mail]
This page is at [http://www.vbapi.com/ref/e/enumfontfamproc.html](http://www.vbapi.com/ref/e/enumfontfamproc.html)
EnumThreadWndProc Callback Function

Public Function EnumThreadWndProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

EnumThreadWndProc is an application-defined callback function used by EnumThreadWindows for each window enumerated. This function should process the window in whatever manner is required. This function does not necessarily have to be named EnumThreadWndProc -- that is merely the name given to it in discussions about the API.

Return Value

If the function returns 0, EnumThreadWindows will immediately stop enumerating thread windows. If the function returns a non-zero value, EnumThreadWindows will continue enumerating thread windows until it can no longer find any.

Visual Basic-Specific Issues

Like all callback functions, EnumThreadWndProc must be declared Public and be defined in a module.

Parameters

hwnd
    A handle to the window which EnumThreadWindows has just found.

lParam
    Whatever value was specified in the parameter list of EnumThreadWindows.

Used By

EnumThreadWindows

Go back to the alphabetical Function listing.
Go back to the Reference section index.

Last Modified: August 15, 1999
EnumWindowsProc Callback Function

Public Function EnumWindowsProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

EnumWindowsProc is an application-defined callback function used by EnumWindows for each window enumerated. This function should process the window in whatever manner is required. This function does not necessarily have to be named EnumWindowsProc -- that is merely the name given to it in discussions about the API.

Return Value

If the function returns 0, EnumWindows will immediately stop enumerating windows. If the function returns a non-zero value, EnumWindows will continue enumerating windows until it can no longer find any.

Visual Basic-Specific Issues

Like all callback functions, EnumWindowsProc must be declared Public and be defined in a module.

Parameters

hwnd
    A handle to the window which EnumWindows has just found.

lParam
    Whatever value was specified in the parameter list of EnumWindows.

Used By

EnumWindows

Go back to the Callback Function listing.
Go back to the Reference section index.
MsgBoxCallback Callback Function

Public Sub MsgBoxCallback (lpHelpInfo As HELPINFO)
    ' application-defined code goes here
End Sub

Description & Usage

The MsgBoxCallback callback function processes help events generated by a message box. It receives information about what context-sensitive help information to display to the user. Like all callback functions, this function does not need to be named MsgBoxCallback inside your program. MsgBoxCallback is merely the name given to it in discussions about the Windows API.

Return Value

MsgBoxCallback does not return a value.

Visual Basic-Specific Issues

Like all callback functions, MsgBoxCallback must be declared Public and be defined in a module.

Parameters

lpHelpInfo
    Identifies what help information to display to the user.

Used By

MSGBOXPARAMS

Back to the Callback Function list.
OFNHookProc Callback Function

Public Function OFNHookProc (ByVal hdlg As Long, ByVal uiMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    ' Application-defined code goes here.
End Function

Description & Usage

The OFNHookProc hook function processes messages for an Open File or Save File common dialog box. This callback function works with file dialogs that have Windows Explorer-type extensions. For file dialogs without these extensions, use OFNHookProcOldStyle instead.

Return Value

If the function returns zero, the default file dialog message handler processes the message. If the function returns a nonzero value, the default file dialog message handler ignores the message.

Visual Basic-Specific Issues

Like most callback functions, this function must be Public and be defined in a module. It does not need to be named OFNHookProc in your program -- that is the name given to it in discussions about the API.

Parameters

hdlg
   A handle to the file dialog's child window. Use GetParent to get a handle to the actual file dialog box.
uiMsg
   The identifier of the message to process.
wParam
   The first message parameter.
lParam
   The second message parameter.

Used By

OPENFILENAME

Back to the Callback Function list.
OFNHookProcOldStyle Callback Function

Public Function OFNHookProcOldStyle (ByVal hdlg As Long, ByVal uiMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    ' Application-defined code goes here.
End Function

Description & Usage

The **OFNHookProcOldStyle** hook function processes messages for an Open File or Save File common dialog box. This callback function works with file dialogs that do not have Windows Explorer-type extensions. For file dialogs with these extensions, use **OFNHookProc** instead.

Return Value

If the function returns zero, the default file dialog message handler processes the message. If the function returns a nonzero value, the default file dialog message handler ignores the message.

Visual Basic-Specific Issues

Like most callback functions, this function must be Public and be defined in a module. It does not need to be named **OFNHookProcOldStyle** in your program -- that is the name given to it in discussions about the API.

Parameters

- **hdlg**
  A handle to the file dialog's child window. Use **GetParent** to get a handle to the actual file dialog box.
- **uiMsg**
  The identifier of the message to process.
- **wParam**
  The first message parameter.
- **lParam**
  The second message parameter.

Used By

**OPENFILENAME**
PropEnumProcEx Callback Function

Public Function PropEnumProcEx (ByVal hwnd As Long, ByVal lpszString As Long, ByVal hData As Long, ByVal dwData As Long) As Long
  ' application-defined code goes here
End Function

Description & Usage

The PropEnumProcEx callback function is used to process an enumerated window property gotten by the EnumPropsEx function. Each window property is passed one at a time to this callback function. This callback function receives not only the property's name but also a handle to its data (whatever the object happens to be). In your program, this function does not actually have to be called PropEnumProcEx -- that is merely the name used for it in discussions regarding the API.

Return Value

If the function returns 0, EnumPropsEx immediately terminates window property enumeration. If the function returns a non-zero value, EnumPropsEx will continue enumeration until the last property is found.

Visual Basic-Specific Issues

Like all callback functions, PropEnumProcEx must be Public and be declared inside a module.

Parameters

hwnd
  A handle to the window whose properties are being enumerated.
lpszString
  A pointer to a string specifying the name of the window property.
hData
  A handle to the window property's data.
dwData
  Additional data specified by EnumPropsEx.

Used By

EnumPropsEx
SoundSentryProc Callback Function

Public Function SoundSentryProc (ByVal dwMillisec As Long, ByVal fwdEffect As Long) As Long
    ' application-defined code goes here
End Function

Description & Usage

SoundSentryProc allows the use of a custom cue used by the SoundSentry accessibility feature. This callback function defines a custom method to use to present some sort of visual cue in place of the ones built into SoundSentry. Unlike most callback functions, this function must be called SoundSentryProc and must be exported by a DLL which is loaded on startup.

Return Value

If the function returns 0, the function failed to display the visual cue. If the function returns a non-zero value, the function displayed the cue successfully.

Visual Basic-Specific Issues

None.

Parameters

*dwMillisec*

The duration in milliseconds over which the visual cue should be displayed.

*fwdEffect*

The type of visual cue to display. The only valid value for this parameter is SSWF_CUSTOM.

Constant Definitions

Const SSWF_CUSTOM = 4

Used By

SOUNDSENTRY
TimerProc Callback Function

Public Sub TimerProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal idEvent As Long, _
ByVal dwTime As Long)
   ' Place application-defined code here.
End Sub

Description & Usage

The TimerProc callback function executes whenever the time-out period of an active timer elapses. Because the callback function receives the index of the timer that called it, a single TimerProc function can be used to handle multiple timers simultaneously.

Return Value

TimerProc does not return a value.

Visual Basic-Specific Issues

Like all callback functions, TimerProc must be Public and be declared inside a module.

Parameters

hwnd
A handle to the window that owns the timer, if any.

uMsg
This always specifies the WM_TIMER message.

idEvent
The unique identifier of the timer that is calling the function.

dwTime
The number of milliseconds that have elapsed since Windows was last started. This is the same as the output from the GetTickCount function.

Constant Definitions

Const WM_TIMER = &H113

Used By
WindowProc Callback Function

Public Function WindowProc (ByVal hwnd As Long, ByVal uMsg As Long, ByVal wParam As Long, ByVal lParam As Long) As Long
' application-defined code goes here
End Function

Description & Usage

The **WindowProc** hook function acts as the window procedure of a window. The window procedure processes all of the messages received by that window. The function does not actually have to be named **WindowProc**; that is just the name given to it in discussions about the API.

Return Value

The return value depends on the requirements of the message being processed.

Visual Basic-Specific Issues

Like all callback functions, **WindowProc** must be declared Public and be defined in a module.

Parameters

- **hwnd**
  A handle to the window which has received the message.
- **uMsg**
  The message to be acted upon.
- **wParam**
  Additional information about the message.
- **lParam**
  Additional information about the message.

Used By

- CallWindowProc, GetClassLong, GetWindowLong, SetClassLong, SetWindowLong, WNDCLASS, WNDCLASSEX

Go back to the Callback Function listing.
Go back to the Reference section index.
Windows API Reference: Macros

Last Update: January 21, 2001
Number of Macros Listed: 15

Below is an alphabetical list of the API macros currently documented on this web site. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

Jump to:  A  |  B  |  C  |  D  |  E  |  F  |  G  |  H  |  I  |  J  |  K  |  L  |  M  |  N  |  O  |  P  |  Q  |  R  |  S  |  T  |  U  |  V  |  W  |  X  |  Y  |  Z

- **F**
  - FIRST_IPADDRESS
  - FOURTH_IPADDRESS

- **G**
  - GET_X_LPARAM
  - GET_Y_LPARAM

- **H**
  - HIBYTE
  - HIWORD

- **L**
  - LOBYTE
  - LOWORD

- **M**
  - MAKELANGID
  - MAKELCID
  - MAKEIPRANGE
  - MAKEPOINTS
  - MAKEWORD

- **S**
- SECOND_IPADDRESS
- T
- THIRD_IPADDRESS

Go back to the Reference section index.
FIRST_IPADDRESS Macro

Public Function FIRST_IPADDRESS (ByVal ipAddress As Long) As Long
    FIRST_IPADDRESS = Val("&H" & Left(Right("00000000" & Hex(ipAddress), 8), 2))
End Function

Description & Usage

FIRST_IPADDRESS extracts the first field (field 0) from an IP address packed inside a 32-bit integer. The IP address must be in host byte order.

Return Value

The macro returns the value of the first field of the IP address.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement FIRST_IPADDRESS this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

ipAddress
The IP address to get the first field of. The IP address must be packed into a 32-bit integer in host byte order.

See Also

FOURTH_IPADDRESS, SECOND_IPADDRESS, THIRD_IPADDRESS
FOURTH_IPADDRESS Macro

Public Function FOURTH_IPADDRESS (ByVal ipAddress As Long) As Long
    FOURTH_IPADDRESS = Val(],&H" & Right("00" & Hex(ipAddress), 2))
End Function

Description & Usage

FOURTH_IPADDRESS extracts the fourth field (field 3) from an IP address packed inside a 32-bit integer. The IP address must be in host byte order.

Return Value

The macro returns the value of the fourth field of the IP address.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement FOURTH_IPADDRESS this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

ipAddress

The IP address to get the fourth field of. The IP address must be packed into a 32-bit integer in host byte order.

See Also

FIRST_IPADDRESS, SECOND_IPADDRESS, THIRD_IPADDRESS
GET_X_LPARAM Macro

Public Function GET_X_LPARAM (ByVal lParam As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(lParam), 8)
    GET_X_LPARAM = CLng("&H" & Right(hexstr, 4))
End Function

Description & Usage

The GET_X_LPARAM macro extracts the x-coordinate from a 32-bit integer packed with a coordinate pair. The macro retrieves the value found in the value's low-order word, which in this case represents the x-coordinate.

Return Value

The macro returns the x-coordinate of the packed coordinate pair.

Visual Basic-Specific Issues

The method I present for extracting the low-order word may seem unusual, since I first convert the value into an 8-digit hex string and then cut that in half. I do this because it is the only way that works 100% of the time in Visual Basic. All the math-based routines I've seen to do this fail when the &H80000000 bit of lParam is set, because VB then sees it as a negative number.

Parameters

lParam

The packed coordinate pair to extract the x-coordinate from.

See Also
GET_Y_LPARAM, MAKEPOINTS

Back to the Macro list.
Back to the Reference section.

Last Modified: July 30, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/g/get_x_lparam.html
GET_Y_LPARAM Macro

Public Function GET_Y_LPARAM (ByVal lParam As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(lParam), 8)
    GET_Y_LPARAM = CLng("&H" & Left(hexstr, 4))
End Function

Description & Usage

The GET_Y_LPARAM macro extracts the y-coordinate from a 32-bit integer packed with a coordinate pair. The macro retrieves the value found in the value's high-order word, which in this case represents the y-coordinate.

Return Value

The macro returns the y-coordinate of the packed coordinate pair.

Visual Basic-Specific Issues

The method I present for extracting the high-order word may seem unusual, since I first convert the value into an 8-digit hex string and then cut that in half. I do this because it is the only way that works 100% of the time in Visual Basic. All the math-based routines I've seen to do this fail when the &H80000000 bit of lParam is set, because VB then sees it as a negative number.

Parameters

lParam

The packed coordinate pair to extract the y-coordinate from.

See Also
GET_X_LPARAM, MAKEPOINTS

Back to the Macro list.
Back to the Reference section.
HIBYTE Macro

Public Function HIBYTE(ByVal wValue As Integer) As Byte
    HIBYTE = Val("&H" & Left(Right("0000" & Hex(wValue), 4), 2))
End Function

Description & Usage

HIBYTE extracts the high-order byte of a 16-bit integer (a word).

Return Value

The macro returns the high-order byte of the value passed to it.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement HIBYTE this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

wValue
    The 16-bit integer to extract the high-order byte of.

See Also

LOBYTE, MAKEWORD

Back to the Macro list.
Back to the Reference section.
HIWORD Macro

Public Function HIWORD (ByVal dwValue As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(dwValue), 8)
    HIWORD = CLng("&H" & Left(hexstr, 4))
End Function

Description & Usage

The HIWORD macro isolates the high-order word from a 32-bit integer.

Return Value

The macro returns the high-order word of the 32-bit integer passed to it.

Visual Basic-Specific Issues

The method I present for extracting the high-order word may seem unusual, since I first convert the value into an 8-digit hex string and then cut that in half. I do this because it is the only way that works 100% of the time in Visual Basic. All the math-based routines I've seen to do this fail when the &H80000000 bit of dwValue is set, because VB then sees it as a negative number.

Parameters

dwValue
    The 32-bit integer to extract the high-order word from.

See Also

LOWORD
LOBYTE Macro

Public Function LOBYTE(ByVal wValue As Integer) As Byte
    LOBYTE = Val("&H" & Right("00" & Hex(wValue), 2))
End Function

Description & Usage

LOBYTE extracts the low-order byte of a 16-bit integer (a word).

Return Value

The macro returns the low-order byte of the value passed to it.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement LOBYTE this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

wValue
The 16-bit integer to extract the low-order byte of.

See Also

HIBYTE, MAKEWORD

Back to the Macro list.
LOWORD Macro

Public Function LOWORD (ByVal dwValue As Long) As Long
    Dim hexstr As String
    hexstr = Right("00000000" & Hex(dwValue), 8)
    LOWORD = CLng("&H" & Right(hexstr, 4))
End Function

Description & Usage

The LOWORD macro isolates the low-order word from a 32-bit integer.

Return Value

The macro returns the low-order word of the 32-bit integer passed to it.

Visual Basic-Specific Issues

The method I present for extracting the low-order word may seem unusual, since I first convert the value into an 8-digit hex string and then cut that in half. I do this because it is the only way that works 100% of the time in Visual Basic. All the math-based routines I've seen to do this fail when the &H80000000 bit of dwValue is set, because VB then sees it as a negative number.

Parameters

\( dwValue \)

The 32-bit integer to extract the low-order word from.

See Also

HIWORD
MAKELANGID Macro

Public Function MAKELANGID (ByVal usPrimaryLanguage As Integer, ByVal usSubLanguage As Integer) As Long
  MAKELANGID = (usSubLanguage * 1024) Or usPrimaryLanguage
End Function

Description & Usage

The MAKELANGID macro creates a language identifier. This language identifier specifies a language using a "primary" language (for example, English, Spanish, French, etc.) and a "sublanguage" or dialect of that language (for example, American English, Australian English, UK English, etc.). Keep in mind that a typical installation of Windows only supports one primary language besides the neutral language, so only a few of the language identifiers generated by MAKELANGID will be supported on any given computer.

Note that the following language/sublanguage combinations have special meanings:

- **LANG_NEUTRAL and SUBLANG_NEUTRAL**: A neutral language, supported by any installation of Windows.
- **LANG_NEUTRAL and SUBLANG_DEFAULT**: The user's default language.
- **LANG_NEUTRAL and SUBLANG_SYS_DEFAULT**: The system's default language.

Return Value

The macro returns the language identifier that represents the specified primary language and sublanguage.

Visual Basic-Specific Issues

None.

Parameters

* **usPrimaryLanguage**
  One LANG_* flag specifying the primary language. Valid flags can be found in the Constant Definitions section below. If using a user-defined primary language, this can be any value between &H200 and &H3FF inclusive.

* **usSubLanguage**
  One SUBLANG_* flag specifying the sublanguage. Valid flags can be found in the Constant Definitions section below. If using a user-defined sublanguage, this can be any value between &H20 and &H3F inclusive.

Constant Definitions
Note: Both the possible flags for \textit{usPrimaryLanguage} and \textit{usSubLanguage} appear below. Since the function of each flag is so obvious given its name, they are presented below without annotation.

\begin{verbatim}
Const LANG_NEUTRAL = &H0
Const LANG_AFRIKAANS = &H36
Const LANG_ALBANIAN = &H1C
Const LANG_ARABIC = &H1
Const LANG_BASQUE = &H2D
Const LANG_BELARUSIAN = &H23
Const LANG_BULGARIAN = &H2
Const LANG_CATALAN = &H3
Const LANG_CHINESE = &H4
Const LANG_CROATIAN = &H1A
Const LANG_CZECH = &H5
Const LANG_DANISH = &H6
Const LANG_DUTCH = &H13
Const LANG_ENGLISH = &H9
Const LANG_ESTONIAN = &H25
Const LANG_FAEROESE = &H38
Const LANG_FARSI = &H29
Const LANG_FINNISH = &HB
Const LANG_FRENCH = &HC
Const LANG_GERMAN = &H7
Const LANG_GREEK = &H8
Const LANG_HEBREW = &HD
Const LANG_HINDI = &H39
Const LANG_HUNGARIAN = &HE
Const LANG_ICELANDIC = &HF
Const LANG_INDONESIAN = &H21
Const LANG_ITALIAN = &H10
Const LANG_JAPANESE = &H11
Const LANG_KOREAN = &H12
Const LANG_LATVIAN = &H26
Const LANG_LITHUANIAN = &H27
Const LANG_MACEDONIAN = &H2F
Const LANG_MALAY = &H3E
Const LANG_NORWEGIAN = &H14
Const LANG_POLISH = &H15
Const LANG_PORTUGUESE = &H16
Const LANG_ROMANIAN = &H18
Const LANG_RUSSIAN = &H19
Const LANG_SERBIAN = &H1A
Const LANG_SLOVAK = &H1B
Const LANG_SLOVENIAN = &H24
Const LANG_SPANISH = &HA
Const LANG_SWAHILI = &H41
Const LANG_SWEDISH = &H1D
Const LANG_THAI = &H1E
Const LANG_TURKISH = &H1F
Const LANG_UKRAINIAN = &H22
Const LANG_VIETNAMESE = &H2A
Const SUBLANG_NEUTRAL = &H0
\end{verbatim}
Const SUBLANG_DEFAULT = &H1
Const SUBLANG_SYS_DEFAULT = &H2
Const SUBLANG_ARABIC = &H1
Const SUBLANG_ARABIC_Iraq = &H2
Const SUBLANG_ARABIC_Egypt = &H3
Const SUBLANG_ARABIC_Libya = &H4
Const SUBLANG_ARABIC_Algeria = &H5
Const SUBLANG_ARABIC_Morocco = &H6
Const SUBLANG_ARABIC_Tunisia = &H7
Const SUBLANG_ARABIC_Oman = &H8
Const SUBLANG_ARABIC_Yemen = &H9
Const SUBLANG_ARABIC_Syria = &HA
Const SUBLANG_ARABIC_Jordan = &HB
Const SUBLANG_ARABIC_Lebanon = &HC
Const SUBLANG_ARABIC_Kuwait = &HD
Const SUBLANG_ARABIC_UAE = &HE
Const SUBLANG_ARABIC_Bahrain = &HF
Const SUBLANG_ARABIC_Qatar = &H10
Const SUBLANG_CHINESE_TRADITIONAL = &H1
Const SUBLANG_CHINESE_SIMPLIFIED = &H2
Const SUBLANG_CHINESE_Hongkong = &H3
Const SUBLANG_CHINESE_Singapore = &H4
Const SUBLANG_DUTCH = &H1
Const SUBLANG_DUTCH_Belgian = &H2
Const SUBLANG_ENGLISH_US = &H1
Const SUBLANG_ENGLISH_UK = &H2
Const SUBLANG_ENGLISH_AUS = &H3
Const SUBLANG_ENGLISH_CAN = &H4
Const SUBLANG_ENGLISH_NZ = &H5
Const SUBLANG_ENGLISH_EIRE = &H6
Const SUBLANG_ENGLISH_SAFRICA = &H7
Const SUBLANG_ENGLISH_JAMAICA = &H8
Const SUBLANG_ENGLISH_CARIBBEAN = &H9
Const SUBLANG_FRENCH = &H1
Const SUBLANG_FRENCH_Belgian = &H2
Const SUBLANG_FRENCH_Canadian = &H3
Const SUBLANG_FRENCH_Swiss = &H4
Const SUBLANG_FRENCH_Luxembourg = &H5
Const SUBLANG_GERMAN = &H1
Const SUBLANG_GERMAN_Swiss = &H2
Const SUBLANG_GERMAN_Austrian = &H3
Const SUBLANG_GERMAN_Luxembourg = &H4
Const SUBLANG_GERMAN_Liechtenstein = &H5
Const SUBLANG_ITALIAN = &H1
Const SUBLANG_ITALIAN_Swiss = &H2
Const SUBLANG_KOREAN = &H1
Const SUBLANG_KOREAN_Johab = &H2
Const SUBLANG_NORWEGIAN_Bokmål = &H1
Const SUBLANG_NORWEGIAN_Nynorsk = &H2
Const SUBLANG_PORTUGUESE = &H2
Const SUBLANG_PORTUGUESE_Brazilian = &H1
Const SUBLANG_SPANISH = &H1
Const SUBLANG_SPANISH_MEXICAN = &H2
Const SUBLANG_SPANISH_MODERN = &H3
Const SUBLANG_SPANISH_GUATEMALA = &H4
Const SUBLANG_SPANISH_COSTARICA = &H5
Const SUBLANG_SPANISH_PANAMA = &H6
Const SUBLANG_SPANISH_DOMINICAN = &H7
Const SUBLANG_SPANISH_VENEZUELA = &H8
Const SUBLANG_SPANISH_COLOMBIA = &H9
Const SUBLANG_SPANISH_PERU = &HA
Const SUBLANG_SPANISH_ARGENTINA = &HB
Const SUBLANG_SPANISH_ECUADOR = &HC
Const SUBLANG_SPANISH_CHILE = &HD
Const SUBLANG_SPANISH_URUGUAY = &HE
Const SUBLANG_SPANISH_PARAGUAY = &HF
Const SUBLANG_SPANISH_BOLIVIA = &H10
MAKELCID Macro

Public Function MAKELCID (ByVal wLanguageId As Integer, ByVal wSortId As Integer) As Long
    MAKELCID = wSortId * &H10000 + wLanguageId
End Function

Description & Usage

The MAKELCID macro creates a locale identifier. This identifier is based on two things: the locale's language and its string sort order.

Return Value

The macro returns the locale identifier that includes the specified language and string sort order.

Visual Basic-Specific Issues

None.

Parameters

wLanguageId
The language identifier of the locale. Use the MAKELANGID macro to generate this value.

wSortId
One of the following flags specifying the string sort order for the locale:
SORT_DEFAULT
    The default sort order.
SORT_CHINESE_BIG5
    The Chinese BIG5 sort order.
SORT_CHINESE_UNICODE
    The Chinese Unicode sort order.
SORT_JAPANESE_XJIS
    The Japanese XJIS sort order.
SORT_JAPANESE_UNICODE
    The Japanese Unicode order.
SORT_KOREAN_KSC
    The Korean KSC sort order.
SORT_KOREAN_UNICODE
    The Korean Unicode sort order.
Constant Definitions

Const SORT_DEFAULT = &H0
Const SORT_CHINESE_BIG5 = &H0
Const SORT_CHINESE_UNICODE = &H1
Const SORT_JAPANESE_XJIS = &H0
Const SORT_JAPANESE_UNICODE = &H1
Const SORT_KOREAN_KSC = &H0
Const SORT_KOREAN_UNICODE = &H1
MAKEIPRANGE Macro

Public Function MAKEIPRANGE (ByVal low As Byte, ByVal high As Byte) As Long
    MAKEIPRANGE = low + CLng(high) * &H100
End Function

Description & Usage

MAKEIPADDRESS packs a lower bound and an upper bound for an IP address field into a single 32-bit integer, suitable for use with the IPM_SETRANGE message.

Return Value

The macro returns the lower and upper bounds of the range packed into a single value.

Visual Basic-Specific Issues

None.

Parameters

low
    The lower bound of the range.

high
    The upper bound of the range.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
MAKEWORD Macro

Public Function MAKEWORD(ByVal bLow As Byte, ByVal bHigh As Byte) As Integer
    MAKEWORD = Val("&H" & Right("00" & Hex(bHigh), 2) & Right("00" & Hex(bLow), 2))
End Function

Description & Usage

MAKEWORD creates a word by concatenating two individual bytes.

Return Value

The macro returns the 16-bit integer generated by the two bytes.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement MAKEWORD this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

bLow
    The low-order byte of the new 16-bit integer.

bHigh
    The high-order byte of the new 16-bit integer.

See Also

HIBYTE, LOBYTE

Back to the Macro list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/m/makeword.html
MAKEPOINTS Macro

Public Function MAKEPOINTS (ByVal lParam As Long) As POINT_TYPE

Description & Usage

The MAKEPOINTS macro converts an (x,y) coordinate pair packed in a single 32-bit integer into the form of a POINT_TYPE structure. The macro separates the values stored in the low and high order words of the packed coordinates and places them into the structure.

Return Value

The macro returns a POINT_TYPE structure holding the unpacked coordinates.

Visual Basic-Specific Issues

None.

Parameters

lParam

The packed coordinate pair to extract both coordinates from.

See Also

GET_X_LPARAM, GET_Y_LPARAM

Back to the Macro list.
Back to the Reference section.
SECOND_IPADDRESS Macro

Public Function SECOND_IPADDRESS (ByVal ipAddress As Long) As Long
    SECOND_IPADDRESS = Val("&H" & Mid(Right("00000000" & Hex(ipAddress), 8), 3, 2))
End Function

Description & Usage

SECOND_IPADDRESS extracts the second field (field 1) from an IP address packed inside a 32-bit integer. The IP address must be in host byte order.

Return Value

The macro returns the value of the second field of the IP address.

Visual Basic-Specific Issues

Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is necessary to implement SECOND_IPADDRESS this way in VB. Other methods to perform this task can fail for large inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an unwanted value.

Parameters

ipAddress

The IP address to get the second field of. The IP address must be packed into a 32-bit integer in host byte order.

See Also

FIRST_IPADDRESS, FOURTH_IPADDRESS, THIRD_IPADDRESS

Last Modified: October 29, 2000
Windows API Guide: THIRD_IPADDRESS Macro

vbapi.com - part of the VB-World Network

THIRD_IPADDRESS Macro
Public Function THIRD_IPADDRESS (ByVal ipAddress As Long) As Long
THIRD_IPADDRESS = Val("&H" & Mid(Right("00000000" & Hex(ipAddress), 8), 5,
2))
End Function

Description & Usage
THIRD_IPADDRESS extracts the third field (field 2) from an IP address packed inside a 32-bit integer. The IP address
must be in host byte order.

Return Value
The macro returns the value of the third field of the IP address.

Visual Basic-Specific Issues
Although it may seem unusual to use string operations for what would otherwise be a mathematical computation, it is
necessary to implement THIRD_IPADDRESS this way in VB. Other methods to perform this task can fail for large
inputs because VB interprets them as negative values, which causes an equivalent mathematical computation to return an
unwanted value.

Parameters
ipAddress
The IP address to get the third field of. The IP address must be packed into a 32-bit integer in host byte order.

See Also
FIRST_IPADDRESS, FOURTH_IPADDRESS, SECOND_IPADDRESS
Back to the Macro list.
Back to the Reference section.

Last Modified: October 29, 2000


Windows API Reference: Other Information

Last Update: January 21, 2001

Below is an alphabetical list of other information related to some of the functions or structures in the Windows API. Most of the following information is used by a group of functions but is too large itself to be placed directly on the corresponding pages. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

- Address Families
- CSIDLs
- Error Codes
- Manufacturer IDs
- MCI Command Strings
- Product IDs
- Virtual Key Codes
- Window Classes & Styles
- Winsock Error Codes

Go back to the Reference section index.

Last Modified: January 21, 2001
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Address Families

Description & Usage

The **address families** specify the different types of protocols that can be used with Winsock functions. The AF_INET family specifies IP-based protocols, which are the ones used on the Internet. However, Winsock supports a variety of other base network protocols you may use instead on other types of networks.

- **AF_12844**
  IEEE 1284.4 WG AF Protocol.
- **AF_APPLETALK**
  Appletalk protocol.
- **AF_ATM**
  Native ATM services protocol.
- **AF_BAN**
  Banyan protocol.
- **AF_CCITT**
  One of the CCITT protocols (such as X.25).
- **AF_CHAOS**
  One of the MIT CHAOS protocols.
- **AF_CLUSTER**
  Microsoft Wolfpack protocol.
- **AF_DATAKIT**
  One of the Datakit protocols.
- **AF_DECnet**
  DECnet protocol.
- **AF_DLI**
  Direct Data Link interface.
- **AF_ECMA**
  A European Computer Manufacturers protocol.
- **AF_FIREFOX**
  A FireFox protocol.
- **AF_HYLINK**
  NSC Hyperchannel protocol.
Windows API Guide: Address Families

AF_IMPLINK
    Arpanet IMP address.

AF_INET
    Internet or other inter-network address (such as UDP/IP or TCP/IP).

AF_INET6
    Internet or other inter-network address using IPv6 addresses.

AF_IPX
    One of the IPX protocols (such as IPX or SPX).

AF_ISO
    One of the ISO protocols.

AF_LAT
    LAT protocol.

AF_NETBIOS
    NetBIOS protocol.

AF_NS
    One of the Xerox NS protocols, including IPX.

AF_OSI
    One of the ISO protocols. (Same as AF_ISO.)

AF_PUP
    A PUP protocol address.

AF_SNA
    IBM SNA protocol.

AF_UNIX
    A Unix-type local-to-host pipe or portal.

AF_UNKNOWN1
    An unknown protocol.

AF_VOICEVIEW
    VoiceView protocol.

Constant Definitions

Const AF_12844 = 25
Const AF_APPLETALK = 16
Const AF_ATM = 22
Const AF_BAN = 21
Const AF_CCITT = 10
Const AF_CHAOS = 5
Const AF_CLUSTER = 24
Const AF_DATAKIT = 9
Const AF_DECnet = 12
Const AF_DLI = 13
Const AF_ECMA = 8
Const AF_FIREFOX = 19
Const AF_HYLINK = 15
Const AF_IMPLINK = 3
Const AF_INET = 2
Const AF_INET6 = 23
Const AF_IPX = 6
Const AF_ISO = 7
Const AF_LAT = 14
Const AF_NETBIOS = 17
Const AF_NS = 6
Const AF_OSI = 7
Const AF_PUP = 4
Const AF_SNA = 11
Const AF_UNIX = 1
Const AF_UNKNOWN1 = 20
Const AF_VOICEVIEW = 18

**Used By**

`gethostbyaddr`, `HOSTENT`

Back to the Reference section.

---

**Last Modified:** December 17, 2000

This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000

Go back to the Windows API Guide home page.

E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at [http://www.vbapi.com/ref/other/addrfam.html](http://www.vbapi.com/ref/other/addrfam.html)
CSIDLS

Description & Usage

A **CSIDL** is a value identifying one of the special folders in the Windows shell. Special folders are folders which the shell uses for special purposes. For example, some special folders are the My Documents folder, the Recycle Bin, and the Internet Cache folder. These special folders can be either a physical path on a disk or a virtual folder. **Windows NT, 2000:** Many of the CSIDLs refer to folders specific to each user. The CSIDL_COMMON_* ones identify the analogous folders shared among all users.

In some cases, a CSIDL can be combined with one of the following flags:

- **CSIDL_FLAG_CREATE**
  - **Windows 2000:** If the folder the CSIDL refers to does not exist, create it.

- **CSIDL_FLAG_DONT_VERIFY**
  - **Windows 2000:** Use the folder the CSIDL refers to even if it does not exist.

Visual Basic-Specific Issues

None.

CSIDLs

- **CSIDL_ADMINTOOLS**
  - **Windows 2000:** The Administration Tools folder.

- **CSIDL_ALTSTARTUP**
  - The non-localized Startup folder.

- **CSIDL_APPDATA**
  - The Application Data folder (used to store common program data).

- **CSIDL_BITBUCKET**
  - The Recycle Bin on the desktop.

- **CSIDL_COMMON_ADMINTOOLS**
Windows 2000: The Administration Tools folder common to all users.

CSIDL_COMMON_APPDATA
Windows 2000: The Application Data folder (used to store common program data) common to all users.

CSIDL_COMMON_ALTSTARTUP
Windows NT, 2000: The non-localized Startup folder common to all users.

CSIDL_COMMON_DESKTOPDIRECTORY
Windows NT, 2000: The Desktop directory (used to store file objects on the Windows desktop) common to all users.

CSIDL_COMMON_DOCUMENTS
Windows NT, 2000: The Documents folder common to all users.

CSIDL_COMMON_FAVORITES
Windows NT, 2000: The Favorites folder common to all users.

CSIDL_COMMON_PROGRAMS
Windows NT, 2000: The Programs folder under the Start Menu common to all users.

CSIDL_COMMON_STARTMENU
Windows NT, 2000: The Start Menu folder common to all users.

CSIDL_COMMON_STARTUP
Windows NT, 2000: The Startup folder under Start Menu\Programs common to all users.

CSIDL_COMMON_TEMPLATES
Windows NT, 2000: The Templates folder common to all users.

CSIDL_CONTROLS
The Control Panel.

CSIDL_COOKIES
The folder used for Internet Explorer's cookie list.

CSIDL_DESKTOP
The Windows desktop.

CSIDL_DESKTOPDIRECTORY
The Desktop directory (used to hold file objects on the Windows desktop).

CSIDL_DRIVES
The My Computer folder.

CSIDL_FAVORITES
The Favorites folder (used primarily to store Internet Explorer's bookmarks).

CSIDL_FONTS
The Fonts directory (used to hold the fonts installed in Windows).

CSIDL_HISTORY
The folder used for Internet Explorer's history list.

CSIDL_INTERNET
The Internet (refering to the Internet Explorer icon on the desktop).

CSIDL_INTERNET_CACHE
The folder used for Internet Explorer's cache.

CSIDL_LOCAL_APPDATA
With Internet Explorer 5.0 or later: Local Application Data folder.
CSIDL_MYPICTURES
   With Internet Explorer 5.0 or later: The My Pictures folder.

CSIDL_NETHOOD
   The Nethood directory (used to hold objects appearing in Network Neighborhood).

CSIDL_NETWORK
   The Network Neighborhood folder.

CSIDL_PERSONAL
   The My Documents folder.

CSIDL_PRINTERS
   The Printers folder (under My Computer).

CSIDL_PRINTHOOD
   The PrintHood directory (used to store printer links).

CSIDL_PROFILE
   With Internet Explorer 5.0 or later: The user profile folder.

CSIDL_PROGRAM_FILES
   With Internet Explorer 5.0 or later: The Program Files folder.

CSIDL_PROGRAM_FILES_COMMON

CSIDL_PROGRAM_FILES_COMMONX86
   Windows 2000: The x86 Common folder under Program Files for RISC systems.

CSIDL_PROGRAM_FILESX86
   Windows 2000: The x86 Program Files directory on RISC systems.

CSIDL_PROGRAMS
   The Programs folder in the Start Menu.

CSIDL_RECENT
   The Recent folder (used for the Documents list in the Start Menu).

CSIDL_SENDO
   The Send To folder (used to store Send To menu items).

CSIDL_STARTMENU
   The Start Menu.

CSIDL_STARTUP
   The Startup folder under Start Menu\Programs.

CSIDL_SYSTEM
   With Internet Explorer 5.0 or later: The Windows System directory.

CSIDL_SYSTEMX86
   Windows 2000: The x86 system directory on RISC systems.

CSIDL_TEMPLATES
   The Templates folder (used to store document templates).

CSIDL_WINDOWS
   With Internet Explorer 5.0 or later: The Windows directory.

Constant Definitions
Const CSIDL_FLAG_CREATE = &H8000
Const CSIDL_FLAG_DONT_VERIFY = &H4000
Const CSIDL_ADMINTOOLS = &H30
Const CSIDL_ALTSTARTUP = &H1D
Const CSIDL_APPDATA = &H1A
Const CSIDL_BITBUCKET = &HA
Const CSIDL_COMMON_ADMINTOOLS = &H2F
Const CSIDL_COMMON_ALTSTARTUP = &H1D
Const CSIDL_COMMON_APPDATA = &H23
Const CSIDL_COMMON_DESKTOPDIRECTORY = &H19
Const CSIDL_COMMON_DOCUMENTS = &H2E
Const CSIDL_COMMON_FAVORITES = &H1F
Const CSIDL_COMMON_PROGRAMS = &H17
Const CSIDL_COMMON_STARTMENU = &H16
Const CSIDL_COMMON_STARTUP = &H18
Const CSIDL_COMMON_TEMPLATES = &H2D
Const CSIDL_CONTROLS = &H3
Const CSIDL_COOKIES = &H21
Const CSIDL_DESKTOP = &H0
Const CSIDL_DESKTOPDIRECTORY = &H10
Const CSIDL_DRIVES = &H11
Const CSIDL_FAVORITES = &H6
Const CSIDL_FONTS = &H14
Const CSIDL_HISTORY = &H22
Const CSIDL_INTERNET = &H1
Const CSIDL_INTERNET_CACHE = &H20
Const CSIDL_LOCAL_APPDATA = &H1C
Const CSIDL_MYPICTURES = &H27
Const CSIDL_NETHOOD = &H13
Const CSIDL_NETWORK = &H12
Const CSIDL_PERSONAL = &H5
Const CSIDL_PRINTER = &H4
Const CSIDL_PRINTERHOD = &H1B
Const CSIDL_PROFILE = &H28
Const CSIDL_PROGRAM_FILES = &H26
Const CSIDL_PROGRAM_FILES_COMMON = &H2B
Const CSIDL_PROGRAM_FILES_COMMONX86 = &H2C
Const CSIDL_PROGRAM_FILESX86 = &H2A
Const CSIDL_PROGRAMS = &H2
Const CSIDL_RECENT = &H8
Const CSIDL_SENDTO = &H9
Const CSIDL_STARTMENU = &HB
Const CSIDL_STARTUP = &H7
Windows API Guide: CSIDLs

Const CSIDL_SYSTEM = &H25
Const CSIDL_SYSTEMX86 = &H29
Const CSIDL_TEMPLATES = &H15
Const CSIDL_WINDOWS = &H24

Used By

SHGetFolderLocation, SHGetFolderPath, SHGetSpecialFolderLocation, SHGetSpecialFolderPath

Go back to the Other Information listing.
Go back to the Reference section index.

Last Modified: September 27, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/clsid.html
Glossary

The following list defines many of the terms used in discussions about the Windows API and especially on this site. Please refer to this glossary whenever you are confused about the meaning of a term. A link to this glossary appears whenever one of the defined terms appears in the Windows API Guide.

Atom
An integer used to identify a string or integer stored in an atom table. Atoms can be stored in either an application's private atom table or the public atom table defined by the operating system.

Brush
A type of bitmap which is used to fill an area. Brushes are usually 8x8 pixel images, forming either a solid color or a pattern.

Callback Function
A function defined by the application which the Windows API will call. Callback functions are usually used whenever the program must process certain information itself at some time in the midst of the execution of an API function. In Visual Basic, all callback functions must be declared Public and be defined in a module.

Caret
The flashing indicator which appears inside of text boxes. The caret identifies where the current position in the text box is. When text is typed, it (usually) appears inserted at the caret's position.

Client Area
The area of a window which contains the "actual" contents of the window. The client area does not include the title bar, the menu bar (if any), any toolbars or status bars, or the window's border. Coordinates measured within a window are usually taken relative to the client area.

Common Dialog
One of the dialog boxes built into the Windows API designed to allow the user to select something. The common dialog boxes include the Open File, Save File, Choose Color, Choose Font, Print, Print Setup, Find Text, and Find & Replace Text boxes.

Current Point
A point on a device also known as the last point referenced. Almost all drawing function ending in -To usually use the current point as the implied point to begin drawing from; they also set the current point to the end point of whatever was drawn. Other drawing functions completely ignore the current point.
Device
An object which performs input or output operations (or both). Devices are used to communicate with the user and include such objects as the keyboard, mouse, and printers. Many windows are also considered to be devices. Devices are usually referred to using a device context. Back

Device Context
An object which both holds information about a device and provides a way to use the device. Device contexts refer to an internal data structure in Windows not accessible to applications; programs can only reference device context via a handle. Back

Dword
A portion of memory, usually a variable, which has a length of four bytes. The term dword is given to anything which is four bytes in length. Literally, a dword is a "double word." Back

File Pointer
A value identifying the current position in an open file. Any synchronous (not overlapped) read or write operations on the file begin at the byte position identified by the file pointer. Back

Handle
A four-byte integer used to identify a wide variety of objects. Handles refer to an internal data structure not accessible to Windows applications which contain information about an object. Back

Hook Function
A special type of callback function. Hook functions are used to process messages associated with an object such as a window or a dialog box. In general, hook functions are called to respond to some sort of message. Back

Information Context
A special type of device context. Unlike device contexts, an information context can only be used to read properties, settings, or other information about a device. It cannot be used to actually do anything with the device. Back

Item Identifier
Identifies some object, usually a folder, used chiefly by the Windows shell. Programs rarely access these directly, but instead refer to them via a pointer. API functions almost always create these as necessary. Back

Locale
A collection of information specifying the type of language used to present certain kinds of information to the user. A locale consists of things such as the language, the date format, the currency format, the time format, etc. None of these settings actually affect the information itself, but how that information is displayed according to the user's preferences. Locales are typically referenced in the API using a numeric identifier. Back

Logical
When used to describe a graphics object (such as a font or brush), this means that the object described is an "ideal" object. Information about the logical object is used by the system to create a physical object which may not exactly match the logical one. However, only physical objects can actually be used. Back

Message
A type of command sent to an object. Physically, a message is identified by a 32-bit integer. Most messages have two parameters associated with them. Most objects continually monitor for newly received messages and act on them. Back

Mutex
A synchronization object used for interprocess communication. A mutex can be owned by at most one thread at any one time, or it can be owned by none. A mutex is in a signaled state when it is unowned by a thread, and in a nonsignaled state when it is owned. Back

Pen
An object which is used to draw lines and curves. Pens define the style, size, and shape of a line or curve. Pens are usually referred to using a handle. Back

Physical
When used to describe a graphics object (such as a font or brush), this means that the object described is an actual object. Physical objects can be used to actually draw on devices. Back

Pointer
A four-byte integer used to identify a physical location in memory (a memory address). Often when using the API in Visual Basic, an implicit pointer to a variable can be used by passing it ByVal (except for Strings, which are always ByVal) as a parameter. Pointers can also be stored in Long-type variables. Back

Region
An object which identifies one or more areas of an area. Each component of a region can be of any shape, including (but not limited to) ellipses, rectangles, and other polygons. Regions can be combined in a variety of ways to create more complicated regions. Back

Semaphore
A synchronization object used for interprocess communication. A semaphore maintains a count between zero and a maximum value. A semaphore is in a signaled state when its count is positive, and in a nonsignaled state when its count is zero. Back

Thread
A path of execution in a program. Each application has at least one thread executing it, although it is possible to have more than one thread in the same instance of a program. Back

Virtual Folder
A folder which is treated like a folder by the Windows shell but does not correspond to a physical directory on any disk. For example, My Computer is a virtual folder which contains all the root drives, the Control Panel virtual folder, and other items. Virtual folders are rarely used by any non-shell functions. Back

Window Class
Defines attributes common to a group of windows. Windows which are created as a member of a certain window class inherit many properties of that class. The window class largely determines the look of its associated windows. Back

Window Property
An additional piece of information attached to a window. Window properties are referenced by a string identifier and hold a handle to the related piece of information. Window properties can be added, read, and removed from a window and do not by default have any bearing on the basic
operations of a window. Back

Word
A portion of memory, usually a variable, which has a length of two bytes. The term word is given to anything which is two bytes in length. Back

Z-order
An internal list identifying the overlapping of windows. Basically, the Z-order determines which windows lie on top of other windows. The top of the Z-order is the window which is on top of all other windows. Back

Go back to the Reference section index.

Last Modified: August 26, 2000
This page is copyright © 2001 Paul Kuliniewicz. Copyright Information
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/glossary.html
Error Codes

The **error codes** are values that identify different types of errors in the Windows API. Most functions do not report the error code directly; instead, they simply return a flag specifying if some error occurred or not. The actual error code can usually be obtained by using the [GetLastError](http://216.26.168.92/vbapi/ref/other/errorcodes.html) function.

The following list identifies various error codes defined in the Windows API. Your application may define its own error codes and use them with the [error functions](http://216.26.168.92/vbapi/ref/other/errorcodes.html). However, make sure that bit 29 (&H20000000) is set in any application-defined error code you define. Because no Windows API-defined error code has that bit set, using that bit assures that your error code will not conflict with an existing one.

**Error Codes**

- **NO_ERROR**
  - No Error / Success.
- **ERROR_SUCCESS**
  - No Error / Success.
- **ERROR_INVALID_FUNCTION**
  - Incorrect function.
- **ERROR_FILE_NOT_FOUND**
  - The system cannot find the file specified.
- **ERROR_PATH_NOT_FOUND**
  - The system cannot find the path specified.
- **ERROR_TOO_MANY_OPEN_FILES**
  - The system cannot open the file because too many files are currently open.
- **ERROR_ACCESS_DENIED**
  - Access denied.
- **ERROR_INVALID_HANDLE**
  - The handle is invalid.
- **ERROR_ARENA_TRASHED**
  - The storage control blocks were destroyed.
- **ERROR_NOTENOUGH_MEMORY**
  - Insufficient memory is available to process the command.
- **ERROR_INVALIDBLOCK**

---

The storage control block address is invalid.

ERROR_BAD_ENVIRONMENT
The environment is incorrect.

ERROR_BAD_FORMAT
An attempt to load a program with an incorrect format was made.

ERROR_INVALID_ACCESS
The access code is invalid.

ERROR_INVALID_DATA
The data are invalid.

ERROR_OUTOFMEMORY
Insufficient memory is available to complete the operation.

ERROR_INVALID_DRIVE
The system cannot find the drive specified.

ERROR_CURRENT_DIRECTORY
The directory cannot be removed.

ERROR_NOTSAMEDEVICE
The system cannot move the file to a different disk drive.

ERROR_NOMOREFILES
There are no more files.

ERROR_WRITEPROTECT
The disk is write protected.

ERROR_BAD_UNIT
The system cannot find the device specified.

ERROR_NOTREADY
The device is not ready.

ERROR_BADCOMMAND
The device does not recognize the command.

ERROR_CRC
Cyclic redundancy check (CRC) data error.

ERROR_BADLENGTH
The length of the issued command is incorrect.

ERRORSEEK
The drive cannot locate a specific area or track on the disk.

ERROR_NODOSDISK
The specified disk cannot be accessed.

ERROR_SECTOR_NOT_FOUND
The drive cannot find the sector requested.

ERROR_OUTOF_PAPER
The printer is out of paper.

ERROR_WRITE_FAULT
The system cannot write to the specified device.

ERROR_READ_FAULT
The system cannot read from the specified device.
ERROR_GEN_FAILURE
   A device attached to the system is not functioning.
ERROR_SHARING_VIOLATION
   The file cannot be accessed because it is in use by another process.
ERROR_LOCK_VIOLATION
   The file cannot be accessed because another process has locked a portion of it.
ERROR_WRONG_DISK
   The wrong disk is in the drive.
ERROR_SHARING_BUFFER_EXCEEDED
   Too many files have been opened for sharing.
ERROR_HANDLE_EOF
   The end of file (EOF) has been reached.
ERROR_HANDLE_DISK_FULL
   The disk is full.
ERROR_NOT_SUPPORTED
   The network request is not supported.
ERROR_REM_NOT_LIST
   The remote computer is not available.
ERROR_DUP_NAME
   A duplicate name exists on the network.
ERROR_BAD_NETPATH
   The network path was not found.
ERROR_NETWORK_BUSY
   The network is busy.
ERROR_DEV_NOT_EXIST
   The specified network resource or device is not available.
ERROR_TOO_MANY_CMDS
   The network BIOS command limit has been reached.
ERROR_ADAP_HDW_ERR
   A network adapter hardware error has occurred.
ERROR_BAD_NET_RESP
   The specified server cannot perform the requested operation.
ERROR_UNEXP_NET_ERROR
   An unexpected network error has occurred.
ERROR_BAD_REM_ADAP
   The remote adapter is incompatible.
ERROR_PRINTQ_FULL
   The printer queue is full.
ERROR_NO_SPOOL_SPACE
   No space to store the file waiting to be printed is available on the server.
ERROR_PRINT_CANCELLED
   The file waiting to be printed was deleted.
ERROR_NETNAME_DELETED
The specified network name is unavailable.
ERROR_NETWORK_ACCESS_DENIED
    Network access is denied.
ERROR_BAD_DEV_TYPE
    The network resource type is incorrect.
ERROR_BAD_NET_NAME
    The network name cannot be found.
ERROR_TOO_MANY_NAMES
    The name limit for the local computer network adapter card was exceeded.
ERROR_TOO_MANY_SESS
    The network BIOS session limit was exceeded.
ERROR_SHARING_PAUSED
    The remote server is paused or is in the process of being started.
ERROR_REQ_NOT_ACCEP
    The network request was not accepted.
ERROR_REDIR_PAUSED
    The specified printer or disk device is paused.
ERROR_FILE_EXISTS
    The file exists.
ERROR_CANT_MAKE
    The directory or file cannot be created.
ERROR_FAIL_I24
    Failure on Interrupt 24 (INT 24).
ERROR_OUT_OF_STRUCTURES
    Storage to process the request is unavailable.
ERROR_ALREADY_ASSIGNED
    The local device name is already in use.
ERROR_INVALID_PASSWORD
    The specified network password is incorrect.
ERROR_INVALID_PARAMETER
    The parameter is incorrect.
ERROR_NET_WRITE_FAULT
    A write fault occurred on the network.
ERROR_NO_PROC_SLOTS
    The system cannot start another process at this time.
ERROR_TOO_MANY_SEMAPHORES
    The system cannot create another semaphore.
ERROR_EXCL_SEM_ALREADY_OWNED
    The exclusive semaphore is already owned by another process.
ERROR_SEM_IS_SET
    The semaphore is set and cannot be closed.
ERROR_TOO_MANY_SEM_REQUESTS
    The semaphore cannot be set again.
ERROR_INVALID_AT_INTERRUPT_TIME
   The system cannot request exclusive semaphores at interrupt time.

ERROR_SEM_OWNER_DIED
   The previous ownership of this semaphore has ended.

ERROR_SEM_USER_LIMIT
   The system has reached the semaphore user limit.

ERROR_DISK_CHANGE
   The program stopped because the alternate disk was not inserted.

ERROR_DRIVE_LOCKED
   The disk is in use or is locked by another process.

ERROR_BROKEN_PIPE
   The pipe has been ended.

ERROR_OPEN_FAILED
   The system cannot open the device or file specified.

ERROR_BUFFER_OVERFLOW
   The file name is too long.

ERROR_DISK_FULL
   There is insufficient space on the disk.

ERROR_NO_MORE_SEARCH_HANDLES
   No more file search handles are available.

ERROR_INVALID_TARGET_HANDLE
   The target file handle is incorrect.

ERROR_INVALID_CATEGORY
   The IOCTL call made by the program is incorrect.

ERROR_INVALID_VERIFY_SWITCH
   The verify-on-write parameter is incorrect.

ERROR_BAD_DRIVER_LEVEL
   The system does not support the command requested.

ERROR_CALL_NOT_IMPLEMENTED
   This function is only valid under Windows NT/2000.

ERROR_SEM_TIMEOUT
   The semaphore timeout experiod has expired.

ERROR_INSUFFICIENT_BUFFER
   The data area passed to a system call is too small.

ERROR_INVALID_NAME
   The syntax of the filename, directory name, or volume label is incorrect.

ERROR_INVALID_LEVEL
   The system call level is incorrect.

ERROR_NO_VOLUME_LABEL
   The disk has no volume label.

ERROR_MOD_NOT_FOUND
   The specified module cannot be found.

ERROR_PROC_NOT_FOUND
The specified procedure cannot be found.

**ERROR_WAIT_NO_CHILDREN**
There are no child processes to wait for.

**ERROR_CHILD_NOT_COMPLETE**
The program cannot run under Windows NT/2000.

**ERROR_DIRECT_ACCESS_HANDLE**
A file handle or open disk partition was attempted to be used for an operation other than raw disk I/O.

**ERROR_NEGATIVE_SEEK**
An attempt was made to move a file pointer before the beginning of the file.

**ERROR_SEEK_ON_DEVICE**
The file pointer cannot be set on the specified device or file.

**ERROR_IS_JOIN_TARGET**
A JOIN or SUBST command cannot be used for a drive that contains previously joined drives.

**ERROR_IS_JOINED**
An attempt was made to use a JOIN or SUBST command on a drive that has already been joined.

**ERROR_IS_SUBSTED**
An attempt was made to use a JOIN or SUBST command on a drive that has already been substituted.

**ERROR_NOT_JOINED**
The system tried to delete the JOIN of a drive that is not joined.

**ERROR_NOT_SUBSTED**
The system tried to delete the SUBST of a drive that is not substituted.

**ERROR_JOIN_TO_JOIN**
The system tried to JOIN a drive to a directory on a joined drive.

**ERROR_SUBST_TO_SUBST**
The system tried to SUBST a drive to a directory on a substituted drive.

**ERROR_JOIN_TO_SUBST**
The system tried to JOIN a drive to a directory on a substituted drive.

**ERROR_SUBST_TO_JOIN**
The system tried to SUBST a drive to a directory on a joined drive.

**ERROR_BUSY_DRIVE**
The system cannot perform a JOIN or SUBST at this time.

**ERRORSAME_DRIVE**
The system cannot JOIN or SUBST a drive to or for a directory on the same drive.

**ERROR_DIR_NOT_ROOT**
The directory is not a subdirectory of the root directory.

**ERROR_DIR_NOT_EMPTY**
The directory is not empty.

**ERROR_IS_SUBST_PATH**
The path specified is being used in a SUBST.

**ERROR_IS_JOIN_PATH**
The path specified is being used in a JOIN.
ERROR_PATH_BUSY
The path specified cannot be used at this time.

ERROR_IS_SUBST_TARGET
An attempt was made to JOIN or SUBST a drive for which a directory on the drive is the target of a previous SUBST.

ERROR_SYSTEM_TRACE
System trace information was not specified in CONFIG.SYS, or tracing is not allowed.

ERROR_INVALID_EVENT_COUNT
The number of specified semaphore events is incorrect.

ERROR_TOO_MANY_MUXWAITERS
DosMuxSemWait did not execute because too many semaphores are already set.

ERROR_INVALID_LIST_FORMAT
The DosMuxSemWait list is incorrect.

ERROR_LABEL_TOO_LONG
The volume label specified is too long and was truncated to 11 characters.

ERROR_TOO_MANY_TCBS
The system cannot create another thread.

ERROR_SIGNAL_REFUSED
The recipient process has refused the signal.

ERROR_DISCARDED
The segment is already discarded and cannot be locked.

ERROR_NOT_LOCKED
The segment is already unlocked.

ERROR_BAD_THREADID_ADDR
The address for the thread ID is incorrect.

ERROR_BAD_ARGUMENTS
The argument string is incorrect.

ERROR_BAD_PATHNAME
The specified path is invalid.

ERROR_SIGNAL_PENDING
A signal is already pending.

ERROR_MAX_THRDS_REACHED
No more threads can be created in the system.

ERROR_LOCK_FAILED
The system was unable to lock a region of a file.

ERROR_BUSY
The requested resource is in use.

ERROR_CANCEL_VIOLATION
A lock request was not outstanding for the supplied cancel region.

ERROR_ATOMIC_LOCKS_NOT_SUPPORTED
The file system does not support atomic changes to the lock type.

ERROR_INVALID_SEGMENT_NUMBER
The system detected an incorrect segment number.
ERROR_INVALID_ORDINAL
   The system cannot run something because of an invalid ordinal.
ERROR_ALREADY_EXISTS
   The system cannot create a file when it already exists.
ERROR_INVALID_FLAG_NUMBER
   An incorrect flag was passed.
ERROR_SEM_NOT_FOUND
   The specified system semaphore name was not found.
ERROR_INVALID_STARTING_CODESEG
   The system cannot run something because of an invalid starting code segment.
ERROR_INVALID_STACKSEG
   The system cannot run something because of an invalid stack segment.
ERROR_INVALID_MODULETYPE
   The system cannot run something because of an invalid module type.
ERROR_INVALID_EXE_SIGNATURE
   The system cannot run something because it cannot run under Windows NT/2000.
ERROR_EXE_MARKED_INVALID
   The system cannot run something because the EXE was marked as invalid.
ERROR_BAD_EXE_FORMAT
   The system cannot run something because it is an invalid Windows NT/2000 application.
ERROR_ITERATED_DATA_EXCEEDS_64K
   The system cannot run something because the iterated data exceed 64KB.
ERROR_INVALID_MALLOCSIZE
   The system cannot run something because of an invalid minimum allocation size.
ERROR_DYNLINK_FROM_INVALID_RING
   The system cannot run something because of a dynalink from an invalid ring.
ERROR_IOPL_NOT_ENABLED
   The system is not presently configured to run this application.
ERROR_INVALID_SEGDPL
   The system cannot run something because of an invalid segment DPL.
ERROR_AUTODATASEG_EXCEEDS_64KB
   The system cannot run something because the automatic data segment exceeds 64K.
ERROR_RING2SEG_MUST_BE_MOVABLE
   The code segment cannot be greater than or equal to 64KB.
ERROR_RELOC_CHAIN_XEEDS_SEGLIM
   The system cannot run something because the reallocation chain exceeds the segment limit.
ERROR_INFLOOP_IN_RELOC_CHAIN
   The system cannot run something because of an infinite loop in the reallocation chain.
ERROR_ENVVAR_NOT_FOUND
   The system could not find the specified environment variable.
ERROR_NO_SIGNAL_SENT
   No process in the command subtree has a signal handler.
ERROR_FILENAME_EXCED_RANGE
The filename or extension is too long.

**ERROR_RING2_STACK_IN_USE**
The ring 2 stack is busy.

**ERROR_META_EXPANSION_TOO_LONG**
The global filename characters (or ?) are entered incorrectly, or too many global filename characters are specified.

**ERROR_INVALID_SIGNAL_NUMBER**
The signal being posted is incorrect.

**ERROR_THREAD_1_INACTIVE**
The signal handler cannot be set.

**ERROR_LOCKED**
The segment is locked and cannot be reallocated.

**ERROR_TOO_MANY_MODULES**
Too many dynamic link modules are attacked to this program or module.

**ERROR_NESTING_NOT_ALLOWED**
Nesting of calls to LoadModule is not allowed.

**ERROR_BAD_PIPE**
The pipe state is invalid.

**ERROR_PIPE_BUSY**
All pipe instances are busy.

**ERROR_NO_DATA**
The pipe is being closed.

**ERROR_PIPE_NOT_CONNECTED**
No process exists on the other end of the pipe.

**ERROR_MORE_DATA**
More data is available.

**ERROR_VC_DISCONNECTED**
The session was cancelled.

**ERROR_INVALID_EA_NAME**
The specified extended attribute name was invalid.

**ERROR_EA_LIST_INCONSISTENT**
The extended attributes are inconsistent.

**ERROR_NO_MORE_ITEMS**
No more data is available.

**ERROR_CANNOT_COPY**
The Copy API cannot be used.

**ERROR_DIRECTORY**
The directory name is invalid.

**ERROR_EAS_DIDNT_FIT**
The extended attributes did not fit into the buffer.

**ERROR_EA_FILE_CORRUPT**
The extended attribute file on the mounted file system is corrupt.

**ERROR_EA_TABLE_FULL**
The extended attribute table file is full.

**ERROR_INVALID_EA_HANDLE**
The specified extended attribute handle is invalid.

**ERROR_EAS_NOT_SUPPORTED**
The mounted file system does not support extended attributes.

**ERROR_NOT_OWNER**
The system attempted to release a mutex not owned by the caller.

**ERROR_TOO_MANY_POSTS**
Too many posts were made to a semaphore.

**ERROR_MR_MID_NOT_FOUND**
The system cannot find the message for the specified message number in the proper message file.

**ERROR_INVALID_ADDRESS**
The system attempted to access an invalid address.

**ERROR_ARITHMETIC_OVERFLOW**
An arithmetic overflow (result > 32 bits) occurred.

**ERROR_PIPE_CONNECTED**
A process already exists on the other end of the pipe.

**ERROR_PIPE_LISTENING**
The system is waiting for a process to open on the other end of the pipe.

**ERROR_EA_ACCESS_DENIED**
Access to the extended attribute was denied.

**ERROR_OPERATED_ABORTED**
The I/O operation has been aborted because of either a thread exit or an application request.

**ERROR_IO_INCOMPLETE**
The overlapped I/O event is not in a signalled state.

**ERROR_IO_PENDING**
The overlapped I/O operation is in progress.

**ERROR_NOACCESS**
An invalid access to a memory location was attempted.

**ERROR_SWAPERROR**
An error performing an inpage operation occurred.

**ERROR_STACK_OVERFLOW**
The stack overflowed because recursion was too deep.

**ERROR_INVALID_MESSAGE**
The window cannot act on the sent message.

**ERROR_CAN_NOT_COMPLETE**
The function cannot be completed.

**ERROR_INVALID_FLAGS**
Invalid flags were used.

**ERROR_UNRECOGNIZED_VOLUME**
The disk volume does not contain a recognized file system.

**ERROR_FILE_INVALID**
The disk volume for a file has been altered such that the opened file is no longer valid.
ERROR_FULLSCREEN_MODE
   The requested operation cannot be performed in full-screen mode.

ERROR_NO_TOKEN
   An attempt to reference a nonexistent token was made.

ERROR_BADDB
   The registry database is corrupt.

ERROR_BADKEY
   The registry key is invalid.

ERROR_CANTOPEN
   The registry key could not be opened.

ERROR_CANTREAD
   The registry key could not be read from.

ERROR_CANTWRITE
   The registry key could not be written to.

ERROR_REGISTRY_RECOVERED
   One of the registry database files was successfully recovered.

ERROR_REGISTRY_CORRUPT
   The registry is corrupt. The cause could be a corrupted registry database file, a corrupted image in system memory, or a failed attempt to recover the registry because of a missing or corrupted log.

ERROR_REGISTRY_IO_FAILED
   An I/O operation initiated by the registry failed unrecoverably.

ERROR_NOT_REGISTRY_FILE
   The system tried to load or restore a file into the registry, but that file is not in the proper file format.

ERROR_KEY_DELETED
   An illegal operation was attempted on a registry key marked for deletion.

ERROR_NO_LOG_SPACE
   The system could not allocate the required space in a registry log.

ERROR_KEY_HAS_CHILDREN
   A symbolic link in a registry key that already has subkeys or values cannot be created.

ERROR_CHILD_MUST_BE_VOLATILE
   A stable subkey cannot be created under a volatile parent key.

ERROR_NOTIFY_ENUM_DIR
   Because a notify change request is being completed and the information is not being returned in the caller's buffer, the caller must now enumerate the files to find the changes.

ERROR_DEPENDENT_SERVICES_RUNNING
   A stop control has been sent to a service which other running services are dependent on.

ERROR_INVALID_SERVICE_CONTROL
   The requested control is not valid for this service.

ERROR_SERVICE_REQUEST_TIMEOUT
   The service did not respond to the start or control request within the timeout period.

ERROR_SERVICE_NO_THREAD
   A thread could not be created for the service.
Windows API Guide: Error Codes

ERROR_SERVICE_DATABASE_LOCKED
The service database is locked.

ERROR_SERVICE_ALREADY_RUNNING
An instance of the service is already running.

ERROR_INVALID_SERVICE_ACCOUNT
The account name for the service is invalid or nonexistent.

ERROR_SERVICE_DISABLED
The specified service is disabled and cannot be started.

ERROR_CIRCULAR_DEPENDENCY
A circular service dependency was specified.

ERROR_SERVICE_DOES_NOT_EXIST
The specified service does not exist.

ERROR_SERVICE_CANNOT_ACCEPT_CTRL
The service cannot accept control messages at this time.

ERROR_SERVICE_NOT_ACTIVE
The service has not been started.

ERROR_FAILED_SERVICE_CONTROLLER_CONNECT
The service process could not connect to the service controller.

ERROR_EXCEPTION_IN_SERVICE
An exception occurred in the service when handling the control request.

ERROR_DATABASE_DOES_NOT_EXIST
The specified database does not exist.

ERROR_SERVICE_SPECIFIC_ERROR
The service has returned a service-specific error code.

ERROR_PROCESS_ABORTED
The process terminated unexpectedly.

ERROR_SERVICE_DEPENDENCY_FAIL
The dependency service or group failed to start.

ERROR_SERVICE_LOGON_FAILED
The service did not start due to a logon failure.

ERROR_SERVICE_START_HANG
After starting, the service hung in a start-pending state.

ERROR_INVALID_SERVICE_LOCK
The specified service database lock is invalid.

ERROR_SERVICE_MARKED_FOR_DELETE
The specified service has been marked for deletion.

ERROR_SERVICE_EXISTS
The specified service already exists.

ERROR_ALREADY_RUNNING_LKG
The system is currently running with the last-known-good configuration.

ERROR_SERVICE_DEPENDENCY_DELETED
The dependency service does not exist or has been marked for deletion.

ERROR_BOOT_ALREADY_ACCEPTED
The current boot has already been accepted for use as the last-known-good control set.

ERROR_SERVICE_NEVER_STARTED
No attempts to start the service have been made.

ERROR_DUPLICATE_SERVICE_NAME
The name is already in use either as a service name or as a service display name.

ERROR_END_OF_MEDIA
The physical end of the tape has been reached.

ERROR_FILEMARK_DETECTED
A tape access reached a filemark.

ERROR_BEGINNING_OF_MEDIA
The beginning of the tape or partition was encountered.

ERROR_SETMARK_DETECTED
A tape access reached the end of a set of files.

ERROR_NO_DATA_DETECTED
No more data is on the tape.

ERROR_PARTITION_FAILURE
The tape could not be partitioned.

ERROR_INVALID_BLOCK_LENGTH
When accessing a new tape of a multivolume partition, the current blocksize is incorrect.

ERROR_DEVICE_NOT_PARTITIONED
The tape partition information could not be found.

ERROR_UNABLE_TO_LOCK_MEDIA
The system was unable to lock the media eject mechanism.

ERROR_UNABLE_TO_UNLOAD_MEDIA
The system was unable to unload the media.

ERROR_MEDIA_CHANGED
The media in the drive may have changed.

ERROR_BUS_RESET
The I/O bus was reset.

ERROR_NO_MEDIA_IN_DRIVE
There is no media in the drive.

ERROR_NO_UNICODE_TRANSLATION
No mapping for the Unicode character exists in the target multi-byte code page.

ERROR_DLL_INIT_FAILED
A dynamic link library initialization routine failed.

ERROR_SHUTDOWN_IN_PROGRESS
A system shutdown is in progress.

ERROR_NO_SHUTDOWN_IN_PROGRESS
The system shutdown could not be aborted because no shutdown is in progress.

ERROR_IO_DEVICE
The request could not be performed because of an device I/O error.

ERROR_SERIAL_NODEVICE
No serial device was successfully initialized; the serial driver will therefore unload.
ERROR_IRQ_BUSY
The system was unable to open a device sharing an interrupt request (IRQ) with other device(s) because at least one of those devices is already opened.

ERROR_MORE_WRITES
A serial I/O operation was completed by another write to the serial port.

ERROR_COUNTER_TIMEOUT
A serial I/O operation completed because the time-out period expired.

ERROR_FLOPPY_ID_MARK_NOT_FOUND
No ID address mark was found on the floppy disk.

ERROR_FLOPPY_WRONG_CYLINDER
A mismatch exists between the floppy disk sector ID field and the floppy disk controller track address.

ERROR_FLOPPY_UNKNOWN_ERROR
The floppy disk controller reported an unrecognized error.

ERROR_FLOPPY_BAD_REGISTERS
The floppy disk controller returned inconsistent results in its registers.

ERROR_DISK_RECALIBRATE_FAILED
While accessing a hard disk, a recalibrate operation failed, even after retries.

ERROR_DISK_OPERATION_FAILED
While accessing a hard disk, a disk operation failed, even after retries.

ERROR_DISK_RESET_FAILED
While accessing a hard disk, a disk controller reset was needed, but even that failed.

ERROR_EOM_OVERFLOW
An EOM overflow occurred.

ERROR_NOT_ENOUGH_SERVER_MEMORY
Not enough server storage is available to process this command.

ERROR_POSSIBLE_DEADLOCK
A potential deadlock condition has been detected.

ERROR_MAPPED_ALIGNMENT
The base address or the file offset specified does not have the proper alignment.

Constant Definitions

Const NO_ERROR = 0
Const ERROR_SUCCESS = 0
Const ERROR_INVALID_FUNCTION = 1
Const ERROR_FILE_NOT_FOUND = 2
Const ERROR_PATH_NOT_FOUND = 3
Const ERROR_TOO_MANY_OPEN_FILES = 4
Const ERROR_ACCESS_DENIED = 5
Const ERROR_INVALID_HANDLE = 6
Const ERROR_ARENA_TRASHED = 7
Const ERROR_NOT_ENOUGH_MEMORY = 8
Const ERROR_INVALID_BLOCK = 9
Const ERROR_BAD_ENVIRONMENT = 10
Const ERROR_BAD_FORMAT = 11
Const ERROR_INVALID_ACCESS = 12
Const ERROR_INVALID_DATA = 13
Const ERROR_OUTOFMEMORY = 14
Const ERROR_INVALID_DRIVE = 15
Const ERROR_CURRENT_DIRECTORY = 16
Const ERROR_NOT_SAME_DEVICE = 17
Const ERROR_NO_MORE_FILES = 18
Const ERROR_WRITE_PROTECT = 19
Const ERROR_BAD_UNIT = 20
Const ERROR_NOT_READY = 21
Const ERROR_BAD_COMMAND = 22
Const ERROR_CRC = 23
Const ERROR_BAD_LENGTH = 24
Const ERROR_SEEK = 25
Const ERROR_NOT_DOS_DISK = 26
Const ERROR_SECTOR_NOT_FOUND = 27
Const ERROR_OUT_OF_PAPER = 28
Const ERROR_WRITE_FAULT = 29
Const ERROR_READ_FAULT = 30
Const ERROR_GEN_FAILURE = 31
Const ERROR_SHARING_VIOLATION = 32
Const ERROR_LOCK_VIOLATION = 33
Const ERROR_WRONG_DISK = 34
Const ERROR_SHARING_BUFFER_EXCEEDED = 36
Const ERROR_HANDLE_EOF = 38
Const ERROR_HANDLE_DISK_FULL = 39
Const ERROR_BAD_NETPATH = 53
Const ERROR_NETWORK_BUSY = 54
Const ERROR_DEV_NOT_EXIST = 55
Const ERROR_TOO_MANY_CMDS = 56
Const ERROR_ADAP_HDW_ERR = 57
Const ERROR_BAD_NET_RESP = 58
Const ERROR_UNEXP_NET_ERR = 59
Const ERROR_BADREM_ADAP = 60
Const ERROR_PRINTQ_FULL = 61
Const ERROR_NO_SPOOL_SPACE = 62
Const ERROR_PRINT_CANCELLED = 63
Const ERROR_NETNAME_DELETED = 64
Const ERROR_NETWORK_ACCESS_DENIED = 65
Const ERROR_BAD_DEV_TYPE = 66
Const ERROR_BAD_NET_NAME = 67
Const ERROR_TOO_MANY_NAMES = 68
Const ERROR_TOO_MANY_SESS = 69
Const ERROR_SHARING_PAUSED = 70
Const ERROR_REQ_NOT_ACEPT = 71
Const ERROR_REDIR_PAUSED = 72
Const ERROR_FILE_EXISTS = 80
Const ERROR_CANNOT_MAKE = 82
Const ERROR_FAIL_I24 = 83
Const ERROR_OUT_OF_STRUCTURES = 84
Const ERROR_ALREADY_ASSIGNED = 85
Const ERROR_INVALID_PASSWORD = 86
Const ERROR_INVALID_PARAMETER = 87
Const ERROR_NET_WRITE_FAULT = 88
Const ERROR_NO_PROC_SLOTS = 89
Const ERROR_TOO_MANY_SEMAPHORES = 100
Const ERROR_EXCL_SEM_ALREADY_OWNED = 101
Const ERROR_SEM_IS_SET = 102
Const ERROR_TOO_MANY_SEM_REQUESTS = 103
Const ERROR_INVALID_AT_INTERRUPT_TIME = 104
Const ERROR_SEM_OWNER_DIED = 105
Const ERROR_SEM_USER_LIMIT = 106
Const ERROR_DISK_CHANGE = 107
Const ERROR_DRIVE_LOCKED = 108
Const ERROR_BROKEN_PIPE = 109
Const ERROR_OPEN_FAILED = 110
Const ERROR_BUFFER_OVERFLOW = 111
Const ERROR_DISK_FULL = 112
Const ERROR_NO_MORE_SEARCH_HANDLES = 113
Const ERROR_INVALID_TARGET_HANDLE = 114
Const ERROR_INVALID_CATEGORY = 117
Const ERROR_INVALID_VERIFY_SWITCH = 118
Const ERROR_BAD_DRIVER_LEVEL = 119
Const ERROR_CALL_NOT_IMPLEMENTED = 120
Const ERROR_SEM_TIMEOUT = 121
Const ERROR_INSUFFICIENT_BUFFER = 122
Const ERROR_INVALID_NAME = 123
Const ERROR_INVALID_LEVEL = 124
Const ERROR_NO_VOLUME_LABEL = 125
Const ERROR_MOD_NOT_FOUND = 126
Const ERROR_PROC_NOT_FOUND = 127
Const ERROR_WAIT_NO_CHILDREN = 128
Const ERROR_CHILD_NOT_COMPLETE = 129
Const ERROR_DIRECT_ACCESS_HANDLE = 130
Const ERROR_NEGATIVE_SEEK = 131
Const ERROR_SEEK_ON_DEVICE = 132
Const ERROR_IS_JOIN_TARGET = 133
Const ERROR_IS_JOINED = 134
Const ERROR_IS_SUBSTED = 135
Const ERROR_NOT_JOINED = 136
Const ERROR_NOT_SUBSTED = 137
Const ERROR_JOIN_TO_JOIN = 138
Const ERROR_SUBST_TO_SUBST = 139
Const ERROR_JOIN_TO_SUBST = 140
Const ERROR_SUBST_TO_JOIN = 141
Const ERROR_BUSY_DRIVE = 142
Const ERRORSAME_DRIVE = 143
Const ERROR_DIR_NOT_ROOT = 144
Const ERROR_DIR_NOT_EMPTY = 145
Const ERROR_IS_SUBST_PATH = 146
Const ERROR_IS_JOIN_PATH = 147
Const ERROR_PATH_BUSY = 148
Const ERROR_IS_SUBST_TARGET = 149
Const ERROR_SYSTEM_TRACE = 150
Const ERROR_INVALID_EVENT_COUNT = 151
Const ERROR_TOO_MANY_MUXWAITERS = 152
Const ERROR_INVALID_LIST_FORMAT = 153
Const ERROR_LABEL_TOO_LONG = 154
Const ERROR_TOO_MANY_TCBS = 155
Const ERROR_SIGNAL_REFUSED = 156
Const ERROR_DISCARDED = 157
Const ERROR_NOT_LOCKED = 158
Const ERROR_BAD_THREADID_ADDR = 159
Const ERROR_BAD_ARGUMENTS = 160
Const ERROR_BAD_PATHNAME = 161
Const ERROR_SIGNAL_PENDING = 162
Const ERROR_MAX_THRDS_REACHED = 164
Const ERROR_LOCK_FAILED = 167
Const ERROR_BUSY = 170
Const ERROR_CANCEL_VIOLATION = 173
Const ERROR_ATOMIC_LOCKS_NOT_SUPPORTED = 174
Const ERROR_INVALID_SEGMENT_NUMBER = 180
Const ERROR_INVALID_ORDINAL = 182
Const ERROR_ALREADY_EXISTS = 183
Const ERROR_INVALID_FLAG_NUMBER = 186
Const ERROR_SEM_NOT_FOUND = 187
Const ERROR_INVALID_STARTING_CODESEG = 188
Const ERROR_INVALID_STACKSEG = 189
Const ERROR_INVALID_MODULETYPE = 190
Const ERROR_INVALID_EXE_SIGNATURE = 191
Const ERROR_BAD_EXE_FORMAT = 193
Const ERROR_ITERATED_DATA_EXCEEDS_64k = 194
Const ERROR_INVALID_MINALLOCSIZE = 195
Const ERROR_DYNLINK_FROM_INVALID_RING = 196
Const ERROR_IOPL_NOT_ENABLED = 197
Const ERROR_INVALID_SEGDPL = 198
Const ERROR_AUTODATASEG_EXCEEDS_64k = 199
Const ERROR_RING2SEG_MUST_BE_MOVABLE = 200
Const ERROR_RELOC_CHAIN_XEEDS_SEGLIM = 201
Const ERROR_INFLOOP_IN_RELOC_CHAIN = 202
Const ERROR_ENVVAR_NOT_FOUND = 203
Const ERROR_NO_SIGNAL_SENT = 205
Const ERROR_FILENAME_EXCED_RANGE = 206
Const ERROR_RING2_STACK_IN_USE = 207
Const ERROR_META_EXPANSION_TOO_LONG = 208
Const ERROR_INVALID_SIGNAL_NUMBER = 209
Const ERROR_THREAD_1_INACTIVE = 210
Const ERROR_LOCKED = 212
Const ERROR_TOO_MANY_MODULES = 214
Const ERROR_NESTING_NOT_ALLOWED = 215
Const ERROR_BAD_PIPE = 230
Const ERROR_PIPE_BUSY = 231
Const ERROR_NO_DATA = 232
Const ERROR_PIPE_NOT_CONNECTED = 233
Const ERROR_MORE_DATA = 234
Const ERROR_VC_DISCONNECTED = 240
Const ERROR_INVALID_EA_NAME = 254
Const ERROR_EA_LIST_INCONSISTENT = 255
Const ERROR_NO_MORE_ITEMS = 259
Const ERROR_CANNOT_COPY = 266
Const ERROR_DIRECTORY = 267
Const ERROR_EAS_DIDNT_FIT = 275
Const ERROR_EA_FILE_CORRUPT = 276
Const ERROR_EA_TABLE_FULL = 277
Const ERROR_INVALID_EA_HANDLE = 278
Const ERROR_EAS_NOT_SUPPORTED = 282
Const ERROR_NOT_OWNER = 288
Const ERROR_TOO_MANY_POSTS = 298
Const ERROR_MR_MID_NOT_FOUND = 317
Const ERROR_INVALID_ADDRESS = 487
Const ERROR_ARITHMETIC_OVERFLOW = 534
Const ERROR_PIPE_CONNECTED = 535
Const ERROR_PIPE_LISTENING = 536
Const ERROR_EA_ACCESS_DENIED = 994
Const ERROR_OPERATION_ABORTED = 995
Const ERROR_IO_INCOMPLETE = 996
Const ERROR_IO_PENDING = 997
Const ERROR_NOACCESS = 998
Const ERROR_SWAPERROR = 999
Const ERROR_STACK_OVERFLOW = 1001
Const ERROR_INVALID_MESSAGE = 1002
Const ERROR_CAN_NOT_COMPLETE = 1003
Const ERROR_INVALID_FLAGS = 1004
Const ERROR_UNRECOGNIZED_VOLUME = 1005
Const ERROR_FILE_INVALID = 1006
Const ERROR_FULLSCREEN_MODE = 1007
Const ERROR_NO_TOKEN = 1008
Const ERROR_BADDB = 1009
Const ERROR_BADKEY = 1010
Const ERROR_CANTOPEN = 1011
Const ERROR_CANTREAD = 1012
Const ERROR_CANTWRITE = 1013
Const ERROR_REGISTRY_RECOVERED = 1014
Const ERROR_REGISTRY_CORRUPT = 1015
Const ERROR_REGISTRY_IO_FAILED = 1016
Const ERROR_NOT_REGISTRY_FILE = 1017
Const ERROR_KEY_DELETED = 1018
Const ERROR_NO_LOG_SPACE = 1019
Const ERROR_KEY_HAS_CHILDREN = 1020
Const ERROR_CHILD_MUST_BE_VOLATILE = 1021
Const ERROR_NOTIFY_ENUM_DIR = 1022
Const ERROR_DEPENDENT_SERVICES_RUNNING = 1051
Const ERROR_INVALID_SERVICE_CONTROL = 1052
Const ERROR_SERVICE_REQUEST_TIMEOUT = 1053
Const ERROR_SERVICE_NO_THREAD = 1054
Const ERROR_SERVICE_DATABASE_LOCKED = 1055
Const ERROR_SERVICE_ALREADY_RUNNING = 1056
Const ERROR_INVALID_SERVICE_ACCOUNT = 1057
Const ERROR_SERVICE_DISABLED = 1058
Const ERROR_CIRCULAR_DEPENDENCY = 1059
Const ERROR_SERVICE_DOES_NOT_EXIST = 1060
Const ERROR_SERVICE_CANNOT_ACCEPT_CTRL = 1061
Const ERROR_SERVICE_NOT_ACTIVE = 1062
Const ERROR_FAILED_SERVICE_CONTROLLER_CONNECT = 1063
Const ERROR_EXCEPTION_IN_SERVICE = 1064
Const ERROR_DATABASE_DOES_NOT_EXIST = 1065
Const ERROR_SERVICE_SPECIFIC_ERROR = 1066
Const ERROR_PROCESS_ABORTED = 1067
Const ERROR_SERVICE_DEPENDENCY_FAIL = 1068
Const ERROR_SERVICE_LOGON_FAILED = 1069
Const ERROR_SERVICE_START_HANG = 1070
Const ERROR_INVALID_SERVICE_LOCK = 1071
Const ERROR_SERVICE_MARKED_FOR_DELETE = 1072
Const ERROR_SERVICE_EXISTS = 1073
Const ERROR_ALREADY_RUNNING_LKG = 1074
Const ERROR_SERVICE_DEPENDENCY_DELETED = 1075
Const ERROR_BOOT_ALREADY_ACCEPTED = 1076
Const ERROR_SERVICE.Never_STARTED = 1077
Const ERROR_DUPLICATE_SERVICE_NAME = 1078
Const ERROR_END_OF_MEDIA = 1079
Const ERROR_FILEMARK_DETECTED = 1080
Const ERROR_BEGINNING_OF_MEDIA = 1081
Const ERROR_SETMARK_DETECTED = 1082
Const ERROR_NO_DATA_DETECTED = 1083
Const ERROR_PARTITION_FAILURE = 1084
Const ERROR_INVALID_BLOCK_LENGTH = 1085
Const ERROR_DEVICE_NOT_PARTITIONED = 1086
Const ERROR_UNABLE_TO_LOCK_MEDIA = 1087
Const ERROR_UNABLE_TO_UNLOAD_MEDIA = 1088
Const ERROR_MEDIA_CHANGED = 1089
Const ERROR_BUS_RESET = 1090
Const ERROR_NO_MEDIA_IN_DRIVE = 1091
Const ERROR_NO_UNICODE_TRANSLATION = 1092
Const ERROR_DLL_INIT_FAILED = 1093
Const ERROR_SHUTDOWN_IN_PROGRESS = 1094
Const ERROR_NO_SHUTDOWN_IN_PROGRESS = 1095
Const ERROR_IO_DEVICE = 1096
Const ERROR_SERIAL_NO_DEVICE = 1097
Const ERROR_IRQ_BUSY = 1098
Const ERROR_MORE_WRITES = 1099
Const ERROR_SHUTDOWN_IN_PROGRESS = 1100
Const ERROR_FILEMARK_DETECTED = 1101
Const ERROR_BEGINNING_OF_MEDIA = 1102
Const ERROR_SETMARK_DETECTED = 1103
Const ERROR_NO_DATA_DETECTED = 1104
Const ERROR_PARTITION_FAILURE = 1105
Const ERROR_INVALID_BLOCK_LENGTH = 1106
Const ERROR_DEVICE_NOT_PARTITIONED = 1107
Const ERROR_UNABLE_TO_LOCK_MEDIA = 1108
Const ERROR_UNABLE_TO_UNLOAD_MEDIA = 1109
Const ERROR_MEDIA_CHANGED = 1110
Const ERROR_BUS_RESET = 1111
Const ERROR_NO_MEDIA_IN_DRIVE = 1112
Const ERROR_NO_UNICODE_TRANSLATION = 1113
Const ERROR_DLL_INIT_FAILED = 1114
Const ERROR_SHUTDOWN_IN_PROGRESS = 1115
Const ERROR_NO_SHUTDOWN_IN_PROGRESS = 1116
Const ERROR_IO_DEVICE = 1117
Const ERROR_SERIAL_NO_DEVICE = 1118
Const ERROR_IRQ_BUSY = 1119
Const ERROR_MORE_WRITES = 1120
Const ERROR_COUNTER_TIMEOUT = 1121
Const ERROR_FLOPPY_ID_MARK_NOT_FOUND = 1122
Const ERROR_FLOPPY_WRONG_CYLINDER = 1123
Const ERROR_FLOPPY_UNKNOWN_ERROR = 1124
Const ERROR_FLOPPY_BAD_REGISTERS = 1125
Const ERROR_DISK_RECALIBRATE_FAILED = 1126
Const ERROR_DISK_OPERATION_FAILED = 1127
Const ERROR_DISK_RESET_FAILED = 1128
Const ERROR_EOM_OVERFLOW = 1129
Const ERROR_NOT_ENOUGH_SERVER_MEMORY = 1130
Const ERROR_POSSIBLE_DEADLOCK = 1131
Const ERROR_MAPPED_ALIGNMENT = 1132

Used By

GetLastError, SetLastError, SetLastErrorEx

Go back to the Other Information listing.
Go back to the Reference section index.

Last Modified: March 19, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/errorcodes.html
Manufacturer IDs

Description & Usage

The manufacturer IDs identify the manufacturer of a device. Normally, manufacturer IDs only apply to multimedia devices such as joysticks or audio devices.

Visual Basic-Specific Issues

None.

Manufacturer IDs

MM_ANTEX
  Antex Electronics Coporation
MM_APPS
  APPS Software International
MM_APT
  Audio Processing Technology
MM_ARTISOFT
  Artisoft, Inc.
MM_AST
  AST Research, Inc.
MM_ATI
  ATI
MM_AUDIOFILE
  Audio, Inc.
MM_AUDIOPT
  Audio Processing Technology
MM_AURAVISION
  AuraVision Corporation
MM_AZTECH
  Aztech Labs, Inc.
MM_CANOPUS
    Canopus, Co., Ltd.
MM_CAT
    Computer Aided Technologies
MM_COMPUSIC
    Compusic
MM_COMPUTER_FRIENDS
    Computer Friends, Inc.
MM_CONTROLRES
    Control Resources Limited
MM_CREATIVE
    Creative Labs, Inc.
MM_DIALOGIC
    Dialogic Corporation
MM_DOLBY
    Dolby Laboratories
MM_DSP_GROUP
    DSP Group, Inc.
MM_DSP_SOLUTIONS
    DSP Solutions, Inc.
MM_ECHO
    Echo Speech Corporation
MM_EPSON
    Seiko Epson Corporation
MM_ESS
    ESS Technology
MM_EVEREX
    Everex Systems, Inc.
MM_EXAN
    EXAN
MM_FUJITSU
    Fujitsu Corp.
MM_GRAVIS
    Advanced Gravis
MM_IBM
    IBM Corporation
MM_ICL_PS
    ICL Personal Systems
MM_ICS
    Integrated Circuit Systems, Inc.
MM_INTEL
    Intel Corporation
MM_INTERACTIVE
InterActive Inc.
MM_IOMAGIC
    I/O Magic Corporation
MM_ITERATEDSYS
    Iterated Systems, Inc.
MM_KORG
    Toshihiko Okuhura, Korg Inc.
MM_LOGITECH
    Logitech, Inc.
MM_LYRRUS
    Lyrrus Inc.
MM_MATSUSHITA
    Matsushita Electronic Industrial Co., Ltd.
MM_MEDIAVISION
    Media Vision, Inc.
MM_MELABS
    microEngineering Labs
MM_METHEUS
    Metheus
MM_MICROSOFT
    Microsoft Corporation.
MM_MOSCOM
    MOSCOM Corporation
MM_MOTOROLA
    Motorola, Inc.
MM_NCR
    NCR Corporation
MM_NEC
    NEC
MM_NEWMEDIA
    New Media Corporation
MM_NMS
    Natural MicroSystems
MM_OKI
    OKI
MM_Olivetti
    Olivetti
MM_OPTI
    OPTi Computers Inc.
MM_ROLAND
    Roland
MM_SCALACS
    SCALACS
MM_SIERRA
    Sierra Semiconductor Corp.
MM_SILICONSOFT
    Silicon Soft, Inc.
MM_SONICFOUNDRY
    Sonic Foundry
MM_SPEECHCOMP
    Speech Compression
MM_SUPERMAC
    Supermac
MM_TANDY
    Tandy Corporation
MM_TRUEVISION
    Truevision
MM_TURTLE_Beach
    Turtle Beach, Inc.
MM_VAL
    Video Associates Labs, Inc.
MM_VIDEOLOGIC
    Videologic
MM_VITEC
    Vitec Multimedia
MM_VOCALTEC
    Vocaltec Ltd.
MM_VOYETRA
    Voyetra
MM_WANGLABS
    Wang Laboratories, Inc.
MM_WILLOWPOND
    Willow Pond Corporation
MM_WINNOV
    Winnov, Inc.
MM_YAMAHAA
    Yamaha Corporation of America

**Constant Definitions**

Const MM_ANTEX = 31
Const MM_APPS = 42
Const MM_APT = 56
Const MM_ARTISOFT = 20
Const MM_AST = 64
Const MM_ATI = 27
Const MM_AUDIOFILE = 47
Const MM_AUDIOOPT = 74
Const MM_AURAVISION = 80
Const MM_AZTECH = 52
Const MM_CANOPUS = 49
Const MM_CAT = 41
Const MM_COMPUSIC = 89
Const MM_COMPUTER_FRIENDS = 45
Const MM_CONTROLRES = 84
Const MM_CREATIVE = 2
Const MM_DIALOGIC = 93
Const MM_DOLBY = 78
Const MM_DSP_GROUP = 43
Const MM_DSP_SOLUTIONS = 25
Const MM_ECHO = 39
Const MM_EPSON = 50
Const MM_ESS = 46
Const MM_EVEREX = 38
Const MM_EXAN = 63
Const MM_FUJITSU = 4
Const MM_GRAVIS = 34
Const MM_IBM = 22
Const MM_ICL_PS = 32
Const MM_ICS = 57
Const MM_INTEL = 33
Const MM_INTERACTIVE = 36
Const MM_IOMAGIC = 82
Const MM_ITERATEDSYS = 58
Const MM_KORG = 55
Const MM_LOGITECH = 60
Const MM_LYRRUS = 88
Const MM_MATSUSHITA = 83
Const MM_MEDIAVISION = 3
Const MM_MELABS = 44
Const MM_METHEUS = 59
Const MM_MICROSOFT = 1
Const MM_MOSCOM = 68
Const MM_MOTOROLA = 48
Const MM_NCR = 62
Const MM_NEC = 26
Const MM_NEWMEDIA = 86
Const MM_NMS = 87
Const MM_OKI = 79
Const MM_OLIVETTI = 81
Const MM_OPTI = 90
Const MM_ROLAND = 24
Const MM_SCALACS = 54
Const MM_SIERRA = 40
Const MM_SILICONSOFT = 69
Const MM_SONICFOUNDRY = 66
Const MM_SPEECHCOMP = 76
Const MM_SUPERMAC = 73
Const MM_TANDY = 29
Const MM_TRUEVISION = 51
Const MM_TURTLE_BEACH = 21
Const MM_VAL = 35
Const MM_VIDEOLOGIC = 53
Const MM_VITEC = 67
Const MM_VOCALTEC = 23
Const MM_VOYETRA = 30
Const MM_WANGLABS = 28
Const MM_WILLOWPOND = 65
Const MM_WINNOV = 61
Const MM_YAMAHA = 37

**Used By**

**AUXCAPS, JOYCAPS, WAVEOUTCAPS**

Go back to the Other Information listing.
Go back to the Reference section index.
MCI Command Strings

Last Update: July 4, 2000
Number of Command Strings Listed: 5 (5 added)

MCI command strings are issued to MCI devices by using the `mciSendString` function. Command strings instruct a Media Control Interface (MCI) device to perform a specific action, such as opening a file, playing a CD, or stopping playback of a video. The MCI, through its command strings, provides a relatively simple way to control multimedia devices configured on the system.

Below is a list of the MCI command strings that currently have information about them in the Windows API Guide. Please keep in mind that this site does not encompass the entire API yet, so unfortunately may not find what you are looking for. To suggest any additions you would like to see made, please contact the author with your request. All pages added since the last update of this site are clearly marked with NEW.

- close NEW
- open NEW
- pause NEW
- play NEW
- stop NEW

Back to Other API Information.
Back to the Reference section.

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/mci/index.html
close MCI Command String

"close lpszDeviceID, lpszFlags"

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The `close` MCI command string closes a device or file previously opened by the `open` command. All resources used by the MCI device or file are freed, unless those resources are shared with other open devices or files.

All MCI devices recognize the `close` command string.

Return Value

The `close` command string does not return any data.

Visual Basic-Specific Issues

None.

Parameters

- **lpszDeviceID**
  The device identifier string of the MCI device or file to close. This may be an alias to the device or file.

- **lpszFlags**
  Zero or more of the following options:
  - "notify"
    When the command finishes, post the `MM_MCINOTIFY` message to the window specified in the call to `mciSendString`.
  - "wait"
    Do not have `mciSendString` return until the command finishes.
Example

To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpszCommand As String, ByVal lpszReturnString As String, ByVal cchReturnLength _
    As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
    fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file. The file C:\Music\canyon.mid
' is opened when the form opens. The Play and Stop buttons behave as you'd expect.
' The only potential surprise is that the current position is not reset when playback
' stops; it
' behaves just as pausing playback would. The file closes when the form unloads.
' If anything goes wrong in the example, display a message box with
' the MCI error message text.

Private Sub Form_Load()
' Open the file "C:\Music\canyon.mid" for later use in the example.
' Give it an alias of "canyon" so we don't need to refer to the filename again.
    Dim errcode As Long ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
' Stop playback of the MIDI file when this button is pressed.
' The position within the file does not move back to the beginning.
    Dim errcode As Long ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)
If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads. This is important, because the
    ' MIDI driver can only work with one file at a time. There's no need to
    ' check
    ' for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code

    errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error.
    ' There's
    ' no reason to use the MessageBox API function; VB's MsgBox function will
    ' suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long    ' return value

    ' Get a string explaining the MCI error.
    errstr = Space(128)
    retval = mciGetErrorString(errcode, errstr, Len(errstr))
    ' Remove the terminating null and empty space at the end.
    errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)

    ' Display a simple error message box.
    retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also

open

Back to the MCI Command list.
Back to Other API Information.
Back to the Reference section.
open MCI Command String

"open lpszDevice, lpszOpenFlags, lpszFlags"

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The `open` MCI command string opens a MCI device, initializing it for further use. One of the most useful features of `open` is the ability to assign an alias to a device, which makes referring to a file opened implicitly through a device simpler. Although in some cases it is not necessary to open a device before using it, it should be done nonetheless. Once your program is finished using the device, it should close it using the `close` command string.

All MCI device types recognize the `open` command string.

Return Value

The `open` command string does not return a value.

Visual Basic-Specific Issues

None.

Parameters

`lpszDevice`

The device identifier string of the MCI device or file to open. When opening a file, it will be manipulated through its typically associated device unless you use the "type" option in `lpszOpenFlags` to use a different device. For example, opening a MIDI file opens it through the sequencer. If this is "new", then a new resource is created; use the "type" option in `lpszOpenFlags` to specify the device type.

`lpszOpenFlags`

Zero or more of the following options for opening the device:

- "elementname" (digitalvideo)
  
  Specifies the file to load when the device opens.
"alias device_alias" (all)
    Assigns an alternate name that can be used in subsequent command strings. This is particularly useful to avoid
    referring to a lengthy filename multiple times. The device_alias can then be used in place of the device ID
    string.
"buffer buffer_size" (waveaudio)
    Sets the size of the waveform audio device buffer, in seconds. Valid values range from 2 to 9 inclusive.
"parent hwnd" (digitalvideo, overlay)
    Specifies a handle to the parent window for any windows the device may open. Windows opened by the device
    will become that window's children.
"shareable" (all)
    Allow this particular instance of the device to be opened multiple times simultaneously, shared among any
    programs that wish to use it. However, the MCISEQ and MCIWAVE drivers do not support sharing.
"style child" (digitalvideo, overlay)
    Open a window with the style of a child window.
"style overlapped" (digitalvideo, overlay)
    Open a window with the style of an overlapped (regular) window.
"style popup" (digitalvideo, overlay)
    Open a window with the style of a pop-up window.
"style style_type" (digitalvideo, overlay)
    Open a window with the specified style. style_type is the numerical value of one of the following flags:
    SW_HIDE
        Hide the window.
    SW_MAXIMIZE
        Maximize the window.
    SW_MINIMIZE
        Minimize the window.
    SW_RESTORE
        Restore the window (not maximized nor minimized).
    SW_SHOW
        Show the window.
    SW_SHOWMAXIMIZED
        Show the window maximized.
    SW_SHOWMINIMIZED
        Show the window minimized.
    SW_SHOWMINNOACTIVE
        Show the window minimized but do not activate it.
    SW_SHOWNA
        Show the window in its current state but do not activate it.
    SW_SHOWNOACTIVATE
        Show the window in its most recent size and position but do not activate it.
    SW_SHOWNORMAL
        Show the window and activate it (as usual).
"type device_type" (all)
    When opening a new instance of a device, this specifies the type of device being opened. Remember that "new"
    was specified as lpszDevice.

lpszFlags
    Zero or more of the following options:
    "notify"
        When the command finishes, post the MM_MCINOTIFY message to the window specified in the call to
        mciSendString.
    "wait"
        Do not have mciSendString return until the command finishes.
Constant Definitions

Const SW_HIDE = 0
Const SW_MAXIMIZE = 3
Const SW_MINIMIZE = 6
Const SW_RESTORE = 9
Const SW_SHOW = 5
Const SW_SHOWMAXIMIZED = 3
Const SW_SHOWMINIMIZED = 2
Const SW_SHOWMINNOACTIVE = 7
Const SW_SHOWNA = 8
Const SW_SHOWNOACTIVATE = 4
Const SW_SHOWNORMAL = 1

Example

To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpszCommand As String, ByVal lpszReturnString As String, ByVal _
    cchReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
    fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file. The file C:\Music\canyon.mid
' is opened when the form opens. The Play and Stop buttons behave as you'd expect. The
' only potential surprise is that the current position is not reset when playback
' stops; it
' behaves just as pausing playback would. The file closes when the form unloads.
' If anything goes wrong in the example, display a message box with
' the MCI error message text.

Private Sub Form_Load()
    ' Open the file "C:\Music\canyon.mid" for later use in the example.
    ' Give it an alias of "canyon" so we don't need to refer to the filename
    again.
    Dim errcode As Long  ' MCI error code

    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
    ' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
    ' Stop playback of the MIDI file when this button is pressed.
    ' The position within the file does not move back to the beginning.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads.  This is important, because the
    ' MIDI driver can only work with one file at a time.  There's no need to
    ' check
    ' for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error.
    ' There's
    ' no reason to use the MessageBox API function; VB's MsgBox function will
    ' suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long    ' return value
    ' Get a string explaining the MCI error.
    errstr = Space(128)
    retval = mciGetErrorString(errcode, errstr, Len(errstr))
    ' Remove the terminating null and empty space at the end.
    errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)
    ' Display a simple error message box.
    retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also

close
pause MCI Command String

"pause lpszDeviceID, lpszFlags"

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The **pause** MCI command string pauses playback of a playing MCI device. Typically, the current position does not change as a result of pausing playback.

The following MCI device types recognize the **stop** command string: cdaudio, digitalvideo, sequencer, vcr, videodisk, waveaudio.

Return Value

The **pause** command string does not return any data.

Visual Basic-Specific Issues

None.

Parameters

`lpszDeviceID`

The device identifier string of the MCI device or file to pause. This may be an alias to the device or file.

`lpszFlags`

Zero or more of the following options:

- "notify"
  
  When the command finishes, post the MM_MCINOTIFY message to the window specified in the call to `mciSendString`.

- "test" (digitalvideo and vcr only)
  
  Test to see if the device supports the **pause** command. The device is not actually paused, and the call to `mciSendString` only returns successfully if the device supports the **pause** command.
Windows API Guide: pause MCI Command String

"wait"

Do not have mciSendString return until the command finishes.

Example

To run this code, place three command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Name another one "cmdStop" and set its Caption to "&Stop MIDI File". Finally, if you haven't figured it out yet, name the last one "cmdPause" and set its Caption to "&Pause MIDI File".

This code is licensed according to the terms and conditions listed here.

Declarations and such needed for the example:
(Copy them to the (declarations) section of a module.)

Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
lpstrCommand As String, ByVal lpstrReturnString As String, ByVal _
cchReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
fdwError As Long, ByVal lpstrErrorText As String, ByVal cchErrorText As Long) As Long

Use the MCI to play, stop, or pause playback of a MIDI file. The file C:\Music\canyon.mid
is opened when the form opens. The various buttons behave as you'd expect. The
only potential surprise is that the current position is not reset when playback stops; it
behaves just as pausing playback would. The file closes when the form unloads.

If anything goes wrong in the example, display a message box with
the MCI error message text.

Private Sub Form_Load()
    ' Open the file "C:\Music\canyon.mid" for later use in the example.
    ' Give it an alias of "canyon" so we don't need to refer to the filename again.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
    ' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
    ' Stop playback of the MIDI file when this button is pressed.

Private Sub cmdPause_Click()
    ' Pause playback of the MIDI file when this button is pressed.
    ' The position within the file, naturally, does not revert to the beginning.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("pause
canyon", "", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads. This is important, because the
    ' MIDI driver can only work with one file at a time. There's no need to
    ' check
    ' for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("close
canyon", "", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error.
    ' There's
    ' no reason to use the MsgBox API function; VB's MsgBox function will
    ' suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long   ' return value
    ' Get a string explaining the MCI error.
    errstr = Space(128)
    retval = mciGetErrorString(errcode, errstr, Len(errstr))
    ' Remove the terminating null and empty space at the end.
    errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)
    ' Display a simple error message box.
    retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also

play, stop

Back to the MCI Command list.
Back to Other API Information.
Back to the Reference section.
play MCI Command String

"play lpszDeviceID, lpszPlayFlags, lpszFlags"

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The **play** MCI command string begins playing a MCI device. By default, playback begins at the current position in the device. Of course, this default can be overridden by options specified in **lpszPlayFlags**.

The following MCI device types recognize the **play** command string: cdaudio, digitalvideo, sequencer, vcr, videodisk, waveaudio.

Return Value

The **play** command string does not return a value.

Visual Basic-Specific Issues

None.

Parameters

**lpszDeviceID**

The device identifier string of the MCI device or file to play. This may be an alias to the device or file.

**lpszPlayFlags**

Zero or more of the following options for how to play the MCI device:

- "at time" (vcr)
  - Specifies when the device should begin playback, especially if it has previously been cued.
- "fast" (videodisc)
  - Play the device at a speed faster than normal.
- "from position" (all)
  - Specifies the position at which to begin playback. If nothing is specified for **position**, playback begins at the
current position.
"fullscreen" (digitalvideo)
   Use a full-screen display. This option only works when playing a compressed file.
"repeat" (digitalvideo)
   Restart playback once the end has been reached.
"reverse" (digitalvideo, vcr, videodisc)
   Play the device backwards. When using this option, the "to" option cannot be used.
"scan" (vcr, videodisc)
   Play as fast as possible without disabling video display, although audio may be disabled. This is essentially fast-forwarding the device.
"slow" (videodisc)
   Play the device at a speed slower than normal.
"speed integer" (videodisc)
   Play the videodisc at integer frames per second.
"to position" (all)
   Specifies the position at which to end playback. If nothing is specified for position, playback ends until the end of the device is reached.
"window" (digitalvideo)
   Use the window associated with the device's instance to display the video.

lpszFlags
Zero or more of the following options:
"notify"
   When the command finishes, post the MM_MCINOTIFY message to the window specified in the call to mciSendString.
"test" (digitalvideo and vcr only)
   Test to see if the device supports the play command. Nothing is played, and the call to mciSendString only returns successfully if the device supports the play command.
"wait"
   Do not have mciSendString return until the command finishes.

Example
To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

' This code is licensed according to the terms and conditions listed here.

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpszCommand As String, ByVal lpszReturnString As String, ByVal _
    cchReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
    fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file. The file C:\Music\canyon.mid is opened when the form opens. The Play and Stop buttons behave as you'd expect.
The only potential surprise is that the current position is not reset when playback stops; it behaves just as pausing playback would. The file closes when the form unloads.

If anything goes wrong in the example, display a message box with the MCI error message text.

Private Sub Form_Load()
    ' Open the file "C:\Music\canyon.mid" for later use in the example.
    ' Give it an alias of "canyon" so we don't need to refer to the filename again.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()
    ' Begin playback of the MIDI file when this button is pressed.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()
    ' Stop playback of the MIDI file when this button is pressed.
    ' The position within the file does not move back to the beginning.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads. This is important, because the MIDI driver can only work with one file at a time. There's no need to check for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error. There's no reason to use the MessageBox API function; VB's MsgBox function will suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long    ' return value

' Get a string explaining the MCI error.
errstr = Space(128)
retval = mciGetErrorString(errcode, errstr, Len(errstr))
' Remove the terminating null and empty space at the end.
errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)

' Display a simple error message box.
retval = MsgBox(errstr, vbOKOnly Or vbCritical)

End Sub

See Also

pause, stop

Back to the MCI Command list.
Back to Other API Information.
Back to the Reference section.
stop MCI Command String

"stop lpszDeviceID, lpszStopFlags, lpszFlags"

Platforms

- **Windows 95**: Supported.
- **Windows 98**: Supported.
- **Windows NT**: Requires Windows NT 3.1 or later.
- **Windows 2000**: Supported.
- **Windows CE**: Not Supported.

Description & Usage

The `stop` MCI command string stops playback of an MCI device. For audio CDs, the current position is reset to the beginning of the track that had been playing. For all other MCI devices, the current position remains where it had been playing.

The following MCI device types recognize the `stop` command string: cdaudio, digitalvideo, sequencer, vcr, videodisk, waveaudio.

Return Value

The `stop` command string does not return any data.

Visual Basic-Specific Issues

None.

Parameters

`lpszDeviceID`

The device identifier string of the MCI device or file to stop playback of. This may be an alias to the device or file.

`lpszStopFlags`

The following option may be specified only for digitalvideo devices:

"hold"

Do not release the resources needed to draw the current still image on the screen. This allows the currently displayed frame of the stopped video to be redrawn as necessary (for example, if the playback window moves).

`lpszFlags`

Zero or more of the following options:

"notify"
When the command finishes, post the `MM_MCINOTIFY` message to the window specified in the call to `mciSendString`.

"wait"

Do not have `mciSendString` return until the command finishes.

**Example**

To run this code, place two command buttons on a form window. Name one "cmdPlay" and set its Caption to "&Play MIDI File". Likewise, name the other one "cmdStop" and set its Caption to "&Stop MIDI File".

' This code is licensed according to the terms and conditions listed [here](http://216.26.168.92/vbapi/ref/other/mci/stop.html).

' Declarations and such needed for the example:
' (Copy them to the (declarations) section of a module.)
Public Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal _
    lpszCommand As String, ByVal lpszReturnString As String, ByVal cchReturnLength As Long, ByVal hwndCallback As Long) As Long
Public Declare Function mciGetErrorString Lib "winmm.dll" Alias "mciGetErrorStringA" (ByVal _
    fdwError As Long, ByVal lpszErrorText As String, ByVal cchErrorText As Long) As Long

' Use the MCI to play or stop playback of a MIDI file. The file C:\Music\canyon.mid is opened when the form opens. The Play and Stop buttons behave as you'd expect. The only potential surprise is that the current position is not reset when playback stops; it behaves just as pausing playback would. The file closes when the form unloads.

' If anything goes wrong in the example, display a message box with the MCI error message text.

Private Sub Form_Load()  
    ' Open the file "C:\Music\canyon.mid" for later use in the example.  
    ' Give it an alias of "canyon" so we don't need to refer to the filename again.  
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("open C:\Music\canyon.mid alias canyon", ",", 0, 0)  
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdPlay_Click()  
    ' Begin playback of the MIDI file when this button is pressed.  
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("play canyon", ",", 0, 0)  
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub cmdStop_Click()  
    ' Stop playback of the MIDI file.  
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)  
    If errcode <> 0 Then DisplayError errcode
End Sub
Private Sub cmdStop_Click()
    ' Stop playback of the MIDI file when this button is pressed.
    ' The position within the file does not move back to the beginning.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("stop canyon", ",", 0, 0)
    If errcode <> 0 Then DisplayError errcode
End Sub

Private Sub Form_Unload(Cancel As Integer)
    ' Close the MIDI file when the form unloads. This is important, because the
    ' MIDI driver can only work with one file at a time. There's no need to
    ' check
    ' for an error here, since we're just closing the file.
    Dim errcode As Long  ' MCI error code
    errcode = mciSendString("close canyon", ",", 0, 0)
End Sub

Private Sub DisplayError(ByVal errcode As Long)
    ' This subroutine displays a dialog box with the text of the MCI error.
    ' There's
    ' no reason to use the MessageBox API function; VB's MsgBox function will
    ' suffice.
    Dim errstr As String  ' MCI error message text
    Dim retval As Long    ' return value
    ' Get a string explaining the MCI error.
    errstr = Space(128)
    retval = mciGetErrorString(errcode, errstr, Len(errstr))
    ' Remove the terminating null and empty space at the end.
    errstr = Left(errstr, InStr(errstr, vbNullChar) - 1)
    ' Display a simple error message box.
    retval = MsgBox(errstr, vbOKOnly Or vbCritical)
End Sub

See Also

pause, play

Back to the MCI Command list.
Back to Other API Information.
Back to the Reference section.

Last Modified: July 4, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
Windows API Guide: stop MCI Command String

E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/mci/stop.html
Product Identifiers

Description & Usage

The product identifiers identify the product name or type of a device. The following flags list the product IDs defined by the Windows API. Note that the manufacturer ID of the device must be known to select the proper device ID. The device IDs below are grouped according to manufacturer. (The manufacturer ID associated with each manufacturer is also shown.)

Visual Basic-Specific Issues

None.

Product Identifiers (Grouped by Manufacturer)

- Audio Processing Technology (MM_APT) Product IDs:
  MM_APT_ACE100CD
  ACE 100 CD.
- Artisoft, Inc. (MM_ARTISOFT) Product IDs:
  MM_ARTISOFT_SBWAVEIN
  Artisoft Sounding Board waveform input.
  MM_ARTISOFT_SBWAVEOUT
  Artisoft Sounding Board waveform output.
- Aztech Labs, Inc. (MM_AZTECH) Product IDs:
  MM_AZTECH_AUX_CD
  Aztech auxiliary CD.
  MM_AZTECH_AUX_LINE
  Aztech auxiliary line input.
  MM_AZTECH_AUX_MIC
  Aztech auxiliary microphone.
  MM_AZTECH_AUX
  Aztech auxiliary.
  MM_AZTECH_DSP16_WAVEIN
Aztech DSP 16 waveform input.
MM_AZTECH_DSP16_WAVEOUT
Aztech DSP 16 waveform output.
MM_AZTECH_DSP16_FMSYNTH
Aztech DSP 16 FM synthesizer.
MM_AZTECH_DSP16_WAVESYNTH
Aztech DSP 16 waveform synthesizer.
MM_AZTECH_NOVA16_WAVEIN
Aztech Nova 16 waveform input.
MM_AZTECH_NOVA16_WAVEOUT
Aztech Nova 16 waveform output.
MM_AZTECH_NOVA16_MIXER
Aztech Nova 16 mixer.
MM_AZTECH_PRO16_WAVEIN
Aztech Pro 16 waveform input.
MM_AZTECH_PRO16_WAVEOUT
Aztech Pro 16 waveform output.
MM_AZTECH_PRO16_FMSYNTH
Aztech Pro 16 FM synthesizer.
MM_AZTECH_WASH16_WAVEIN
Aztech Wash 16 waveform input.
MM_AZTECH_WASH16_WAVEOUT
Aztech Wash 16 waveform output.
MM_AZTECH_WASH16_MIXER
Aztech Wash 16 mixer.
MM_AZTECH_MIDIOUT
Aztech MIDI output.
MM_AZTECH_MIDIIN
Aztech MIDI input.
MM_AZTECH_WAVEIN
Aztech waveform input.
MM_AZTECH_WAVEOUT
Aztech waveform output.
MM_AZTECH_FMSYNTH
Aztech FM synthesizer.
MM_AZTECH_MIXER
Aztech mixer.

- **Computer Aided Technologies (MM_CAT) Product IDs:**
  MM_CAT_WAVEOUT
  Waveform output.

- **Creative Labs, Inc. (MM_CREATIVE) Product IDs:**
  MM_CREATIVE_AUX_CD
  Sound Blaster Pro auxiliary CD.
MM_CREATIVE_AUX_LINE
Sound Blaster Pro auxiliary line input.

MM_CREATIVE_AUX_MIC
Sound Blaster Pro auxiliary microphone.

MM_CREATIVE_AUX_MASTER
Sound Blaster Pro auxiliary master.

MM_CREATIVE_AUX_PCSPK
Sound Blaster Pro auxiliary PC speaker.

MM_CREATIVE_AUX_WAVE
Sound Blaster Pro auxiliary waveform.

MM_CREATIVE_AUX_MIDI
Sound Blaster Pro auxiliary MIDI.

MM_CREATIVE_SB15_WAVEIN
Sound Blaster 1.5 waveform input.

MM_CREATIVE_SB15_WAVEOUT
Sound Blaster 1.5 waveform output.

MM_CREATIVE_SB16_MIXER
Sound Blaster Pro 16 mixer.

MM_CREATIVE_SB20_WAVEIN
Sound Blaster 2.0 waveform input.

MM_CREATIVE_SB20_WAVEOUT
Sound Blaster 2.0 waveform output.

MM_CREATIVE_SBP16_WAVEIN
Sound Blaster Pro 16 waveform input.

MM_CREATIVE_SBP16_WAVEOUT
Sound Blaster Pro 16 waveform output.

MM_CREATIVE_SBPRO_WAVEIN
Sound Blaster Pro waveform input.

MM_CREATIVE_SBPRO_WAVEOUT
Sound Blaster Pro waveform output.

MM_CREATIVE_SBPRO_MIXER
Sound Blaster Pro mixer.

MM_CREATIVE_MIDIOUT
Sound Blaster MIDI output.

MM_CREATIVE_MIDIIN
Sound Blaster MIDI input.

MM_CREATIVE_FMSYNTH_MONO
Sound Blaster mono FM synthesizer.

MM_CREATIVE_FMSYNTH_STEREO
Sound Blaster stereo FM synthesizer.

MM_CREATIVE_MIDI_AWE32
Sound Blaster MIDI AWE 32.

- **DSP Group, Inc. (MM_DSP_GROUP) Product IDs:**
MM_DSP_GROUP_TRUESPEECH
True Speech.

- DSP Solutions, Inc. (MM_DSP_SOLUTIONS) Product IDs:
  - MM_DSP_SOLUTIONS_WAVEOUT
    DSP Solutions waveform output.
  - MM_DSP_SOLUTIONS_WAVEIN
    DSP Solutions waveform input.
  - MM_DSP_SOLUTIONS_SYNTH
    DSP Solutions synthesizer.
  - MM_DSP_SOLUTIONS_AUX
    DSP Solutions auxiliary.

- Echo Speech Corporation (MM_ECHO) Product IDs:
  - MM_ECHO_SYNTH
    Echo synthesizer.
  - MM_ECHO_WAVEOUT
    Echo waveform output.
  - MM_ECHO_WAVEIN
    Echo waveform input.
  - MM_ECHO_MIDIOUT
    Echo MIDI output.
  - MM_ECHO_MIDIIN
    Echo MIDI input.
  - MM_ECHO_AUX
    Echo auxiliary.

- ESS Technology (MM_ESS) Product IDs:
  - MM_ESS_AMWAVEOUT
    AM waveform output.
  - MM_ESS_AMWAVEIN
    AM waveform input.
  - MM_ESS_AMAUX
    AM auxiliary.
  - MM_ESS_AMSYNTH
    AM synthesizer.
  - MM_ESS_AMMIDIOUT
    AM MIDI output.
  - MM_ESS_AMMIDIIN
    AM MIDI input.
  - MM_ESS_AUX_CD
    Auxiliary CD.
  - MM_ESS_ES488_WAVEOUT
    ES488 waveform output.
  - MM_ESS_ES488_WAVEIN
    ES488 waveform input.
MM_ESS_ES488_MIXER
ES488 mixer.

MM_ESS_ES688_WAVEOUT
ES688 waveform output.

MM_ESS_ES688_WAVEIN
ES488 waveform input.

MM_ESS_ES688_MIXER
ES488 mixer.

MM_ESS_ES1488_WAVEOUT
ES1488 waveform output.

MM_ESS_ES1488_WAVEIN
ES1488 waveform input.

MM_ESS_ES1488_MIXER
ES1488 mixer.

MM_ESS_ES1688_WAVEOUT
ES1688 waveform output.

MM_ESS_ES1688_WAVEIN
ES1688 waveform input.

MM_ESS_ES1688_MIXER
ES1688 mixer.

MM_ESS_MPU401_MIDIOUT
MPU401 MIDI output.

MM_ESS_MPU401_MIDIIN
MPU401 MIDI input.

MM_ESS_MPU401_MIDIOUT
MPU401 MIDI output.

MM_ESS_MPU401_MIDIIN
MPU401 MIDI input.

MM_ESS_MPU401_MIDIOUT
MPU401 MIDI output.

MM_ESS_MPU401_MIDIIN
MPU401 MIDI input.

Everex Systems, Inc. (MM_EVEREX) Product IDs:
MM_EVEREX_CARRIER
Everex Carrier.

IBM Corporation (MM.ibm) Product IDs:
MM_IBM_PCMCIA_WAVEIN
IBM PCMCIA waveform input.

MM_IBM_PCMCIA_WAVEOUT
IBM PCMCIA waveform output.

MM_IBM_PCMCIA_SYNTH
IBM PCMCIA synthesizer.

MM_IBM_PCMCIA_MIDIIN
IBM PCMCIA MIDI input.

MM_IBM_PCMCIA_MIDIOUT
IBM PCMCIA MIDI output.

MM_IBM_PCMCIA_AUX
IBM PCMCIA auxiliary.

MM_MMOTION_WAVEAUX
IBM M-Motion auxiliary.
MM_MMOTION_WAVEOUT
IBM M-Motion waveform output.
MM_MMOTION_WAVEIN
IBM M-Motion waveform input.

- **Integrated Circuit Systems, Inc. (MM_ICS) Product IDs:**
  
  - MM_ICS_WAVEDECK_WAVEOUT
    Wavedeck waveform output.
  
  - MM_ICS_WAVEDECK_WAVEIN
    Wavedeck waveform input.
  
  - MM_ICS_WAVEDECK_MIXER
    Wavedeck mixer.
  
  - MM_ICS_WAVEDECK_AUX
    Wavedeck auxiliary.
  
  - MM_ICS_WAVEDECK_SYNTH
    Wavedeck synthesizer.

- **InterActive, Inc. (MM_INTERACTIVE) Product IDs:**
  
  - MM_INTERACTIVE_WAVEIN
    InterActive waveform input.
  
  - MM_INTERACTIVE_WAVEOUT
    InterActive waveform output.

- **I/O Magic Corporation (MM_IOMAGIC) Product IDs:**
  
  - MM_IOMAGIC_TEMPO_WAVEOUT
    Tempo waveform output.
  
  - MM_IOMAGIC_TEMPO_WAVEIN
    Tempo waveform input.
  
  - MM_IOMAGIC_TEMPO_SYNTH
    Tempo synthesizer.
  
  - MM_IOMAGIC_TEMPO_MIDIOUT
    Tempo MIDI output.
  
  - MM_IOMAGIC_TEMPO_MXDOUT
    Tempo mixed output.
  
  - MM_IOMAGIC_TEMPO_AUXOUT
    Tempo auxiliary output.

- **Iterated Systems, Inc. (MM_ITERATEDSYS) Product IDs:**
  
  - MM_ITERATED_SYS_FUFCODEC
    FUF Codec.

- **Toshihiko Okuhura, Korg Inc. (MM_KORG) Product IDs:**
  
  - MM_KORG_PCIF_MIDIOUT
    PCIF MIDI output.
  
  - MM_KORG_PCIF_MIDIIN
    PCIF MIDI input.

- **Lyrrus, Inc. (MM_LYRRUS) Product IDs:**
MM_LYRRUS_BRIDGE_GUITAR
Bridge Guitar MIDI.

- **Matsushita Electronic Industrial Co., Ltd. (MM_MATSUSHITA) Product IDs:**
  MM_MATSUSHITA_WAVEIN
  Matsushita waveform input.
  MM_MATSUSHITA_WAVEOUT
  Matsushita waveform output.
  MM_MATSUSHITA_FMSYNTH_STEREO
  Matsushita FM stereo synthesizer.
  MM_MATSUSHITA_MIXER
  Matsushita mixer.
  MM_MATSUSHITA_AUX
  Matsushita auxiliary.

- **Media Vision, Inc. (MM_MEDIAVISION) Product IDs:**
  MM_CDPC_MIDIOUT
  CDPC MIDI output.
  MM_CDPC_MIDIIN
  CDPC MIDI input.
  MM_CDPC_SYNTH
  CDPC synthesizer.
  MM_CDPC_WAVEOUT
  CDPC waveform output.
  MM_CDPC_WAVEIN
  CDPC waveform input.
  MM_CDPC_MIXER
  CDPC mixer.
  MM_CDPC_AUX
  CDPC auxiliary.
  MM_OPUS1208_MIDIOUT
  Opus MV 1208 Chipset MIDI output.
  MM_OPUS1208_MIDIIN
  Opus MV 1208 Chipset MIDI input.
  MM_OPUS1208_SYNTH
  Opus MV 1208 Chipset synthesizer.
  MM_OPUS1208_WAVEOUT
  Opus MV 1208 Chipset waveform output.
  MM_OPUS1208_WAVEIN
  Opus MV 1208 Chipset waveform input.
  MM_OPUS1208_MIXER
  Opus MV 1208 Chipset mixer.
  MM_OPUS1208_AUX
  Opus MV 1208 Chipset auxiliary.
Opus MV 1216 Chipset MIDI output.

MM_OPUS1216_MIDIIN
Opus MV 1216 Chipset MIDI input.

MM_OPUS1216_SYNTH
Opus MV 1216 Chipset synthesizer.

MM_OPUS1216_WAVEOUT
Opus MV 1216 Chipset waveform output.

MM_OPUS1216_WAVEIN
Opus MV 1216 Chipset waveform input.

MM_OPUS1216_MIXER
Opus MV 1216 Chipset mixer.

MM_OPUS1216_AUX
Opus MV 1216 Chipset auxiliary.

MM_PROAUD_MIDIOUT
Pro Audio Spectrum MIDI output.

MM_PROAUD_MIDIIN
Pro Audio Spectrum MIDI input.

MM_PROAUD_SYNTH
Pro Audio Spectrum synthesizer.

MM_PROAUD_WAVEOUT
Pro Audio Spectrum waveform output.

MM_PROAUD_WAVEIN
Pro Audio Spectrum waveform input.

MM_PROAUD_MIXER
Pro Audio Spectrum mixer.

MM_PROAUD_AUX
Pro Audio Spectrum auxiliary.

MM_PROAUD_16_MIDIOUT
Pro Audio Spectrum 16 MIDI output.

MM_PROAUD_16_MIDIIN
Pro Audio Spectrum 16 MIDI input.

MM_PROAUD_16_SYNTH
Pro Audio Spectrum 16 synthesizer.

MM_PROAUD_16_WAVEOUT
Pro Audio Spectrum 16 waveform output.

MM_PROAUD_16_WAVEIN
Pro Audio Spectrum 16 waveform input.

MM_PROAUD_16_MIXER
Pro Audio Spectrum 16 mixer.

MM_PROAUD_16_AUX
Pro Audio Spectrum 16 auxiliary.

MM_PROAUD_PLUS_MIDIOUT
Pro Audio Spectrum Plus MIDI output.
Windows API Guide: Product Identifiers

MM_PROAUD_PLUS_MIDIIN
    Pro Audio Spectrum Plus MIDI input.

MM_PROAUD_PLUS_SYNTH
    Pro Audio Spectrum Plus synthesizer.

MM_PROAUD_PLUS_WAVEOUT
    Pro Audio Spectrum Plus waveform output.

MM_PROAUD_PLUS_WAVEIN
    Pro Audio Spectrum Plus waveform input.

MM_PROAUD_PLUS_MIXER
    Pro Audio Spectrum Plus mixer.

MM_PROAUD_PLUS_AUX
    Pro Audio Spectrum Plus auxiliary.

MM_STUDIO_16_MIDIOUT
    Pro Audio Studio 16 MIDI output.

MM_STUDIO_16_MIDIIN
    Pro Audio Studio 16 MIDI input.

MM_STUDIO_16_SYNTH
    Pro Audio Studio 16 synthesizer.

MM_STUDIO_16_WAVEOUT
    Pro Audio Studio 16 waveform output.

MM_STUDIO_16_WAVEIN
    Pro Audio Studio 16 waveform input.

MM_STUDIO_16_MIXER
    Pro Audio Studio 16 mixer.

MM_STUDIO_16_AUX
    Pro Audio Studio 16 auxiliary.

MM_THUNDER_SYNTH
    Thunder Board synthesizer.

MM_THUNDER_WAVEOUT
    Thunder Board waveform output.

MM_THUNDER_WAVEIN
    Thunder Board waveform input.

MM_THUNDER_AUX
    Thunder Board auxiliary.

MM_TPORT_WAVEOUT
    Audio Port waveform output.

MM_TPORT_WAVEIN
    Audio Port waveform input.

MM_TPORT_SYNTH
    Audio Port synthesizer.

- **microEngineering Labs (MM_MELABS) Product IDs:**
  MM_MELABS_MIDI2GO
    MIDI 2 Go.
- **Metheus (MM_METHEUS) Product IDs:**
  - MM_METHEUS_ZIPPER
    - Zipper.
- **Microsoft (MM_MICROSOFT) Product IDs:**
  - MM_MIDI_MAPPER
    - MIDI mapper.
  - MM_WAVE_MAPPER
    - Waveform mapper.
  - MM_SNDBLST_MIDIOUT
    - Sound Blaster MIDI output port.
  - MM_SNDBLST_MIDIIN
    - Sound Blaster MIDI input port.
  - MM_SNDBLST_SYNTH
    - Sound Blaster internal synthesizer.
  - MM_SNDBLST_WAVEOUT
    - Sound Blaster waveform output.
  - MM_SNDBLST_WAVEIN
    - Sound Blaster waveform input.
  - MM_ADLIB
    - Ad Lib-compatible synthesizer.
  - MM_MPU401_MIDIOUT
    - MPU 401-compatible MIDI output port.
  - MM_MPU401_MIDIIN
    - MPU 401-compatible MIDI input port.
  - MM_PC_JOYSTICK
    - Joystick adapter.
  - MM_PCSPEAKER_WAVEOUT
    - PC Speaker waveform output.
  - MM_MSFT_WSS_WAVEIN
    - Microsoft Audio Board waveform input.
  - MM_MSFT_WSS_WAVEOUT
    - Microsoft Audio Board waveform output.
  - MM_MSFT_WSS_FMSYNTH_STEREO
    - Microsoft Audio Board stereo FM synthesizer.
  - MM_MSFT_WSS_MIXER
    - Microsoft Audio Board mixer driver.
  - MM_MSFT_WSS_OEM_WAVEIN
    - Microsoft OEM Audio Board waveform input.
  - MM_MSFT_WSS_OEM_WAVEOUT
    - Microsoft OEM Audio Board waveform output.
  - MM_MSFT_WSS_OEM_FMSYNTH_STEREO
    - Microsoft OEM Audio Board stereo FM synthesizer.
Microsoft Audio Board auxiliary port.
-MM_MSFT_WSS_OEM_AUX
Microsoft OEM Audio Board auxiliary port.
-MM_MSFT_GENERIC_WAVEIN
Microsoft Vanilla driver waveform input.
-MM_MSFT_GENERIC_WAVEOUT
Microsoft Vanilla driver waveform output.
-MM_MSFT_GENERIC_MIDIIN
Microsoft Vanilla driver MIDI input.
-MM_MSFT_GENERIC_MIDIOUT
Microsoft Vanilla driver MIDI output.
-MM_MSFT_GENERIC_MIDISYNTH
Microsoft Vanilla driver MIDI synthesizer.
-MM_MSFT_GENERIC_AUX_LINE
Microsoft Vanilla driver auxiliary line input.
-MM_MSFT_GENERIC_AUX_MIC
Microsoft Vanilla driver auxiliary microphone.
-MM_MSFT_GENERIC_AUX_CD
Microsoft Vanilla driver auxiliary CD.
-MM_MSFT_WSS_OEM_MIXER
Microsoft OEM Audio Board mixer driver.
-MM_MSFT_MSACM
Microsoft Audio Compression Manager.
-MM_MSFT_ACM_MSADPCM
Microsoft ADPCM Codec.
-MM_MSFT_ACM_IMAADPCM
IMA ADPCM Codec.
-MM_MSFT_ACM_MSFILTER
Microsoft Filter.
-MM_MSFT_ACM_GSM610
GSM 610 Codec.
-MM_MSFT_ACM_G711
G.711 Codec.
-MM_MSFT_ACM_PCM
PCM Converter.
-MM_WSS_SB16_WAVEIN
Sound Blaster 16 waveform input.
-MM_WSS_SB16_WAVEOUT
Sound Blaster 16 waveform output.
-MM_WSS_SB16_MIDIIN
Sound Blaster 16 MIDI input.
-MM_WSS_SB16_MIDIOUT
Sound Blaster 16 MIDI output.
MM_WSS_SB16_SYNTH
    Sound Blaster 16 FM synthesizer.
MM_WSS_SB16_AUX_LINE
    Sound Blaster 16 auxiliary line input.
MM_WSS_SB16_AUX_CD
    Sound Blaster 16 auxiliary CD.
MM_WSS_SB16_MIXER
    Sound Blaster 16 mixer device.
MM_WSS_SBPRO_WAVEIN
    Sound Blaster Pro waveform input.
MM_WSS_SBPRO_WAVEOUT
    Sound Blaster Pro waveform output.
MM_WSS_SBPRO_MIDIIN
    Sound Blaster Pro MIDI input.
MM_WSS_SBPRO_MIDIOUT
    Sound Blaster Pro MIDI output.
MM_WSS_SBPRO_SYNTH
    Sound Blaster Pro FM synthesizer.
MM_WSS_SBPRO_AUX_LINE
    Sound Blaster Pro auxiliary line input.
MM_WSS_SBPRO_AUX_CD
    Sound Blaster Pro auxiliary CD.
MM_WSS_SBPRO_MIXER
    Sound Blaster Pro mixer.
MM_MSFT_WSS_NT_WAVEIN
    WSS NT waveform input.
MM_MSFT_WSS_NT_WAVEOUT
    WSS NT waveform output.
MM_MSFT_WSS_NT_FMSYNTH_STEREO
    WSS NT FM synthesizer.
MM_MSFT_WSS_NT_MIXER
    WSS NT mixer.
MM_MSFT_WSS_NT_AUX
    WSS NT auxiliary.
MM_MSFT_SB16_WAVEIN
    Sound Blaster 16 waveform input.
MM_MSFT_SB16_WAVEOUT
    Sound Blaster 16 waveform output.
MM_MSFT_SB16_MIDIIN
    Sound Blaster 16 MIDI input.
MM_MSFT_SB16_MIDIOUT
    Sound Blaster 16 MIDI output.
MM_MSFT_SB16_SYNTH
Sound Blaster 16 FM synthesizer.
MM_MSFT_SB16_AUX_LINE
Sound Blaster 16 auxiliary line input.
MM_MSFT_SB16_AUX_CD
Sound Blaster 16 auxiliary CD.
MM_MSFT_SB16_MIXER
Sound Blaster 16 mixer device.
MM_MSFT_SBPRO_WAVEIN
Sound Blaster Pro waveform input.
MM_MSFT_SBPRO_WAVEOUT
Sound Blaster Pro waveform output.
MM_MSFT_SBPRO_MIDIIN
Sound Blaster Pro MIDI input.
MM_MSFT_SBPRO_MIDIOUT
Sound Blaster Pro MIDI output.
MM_MSFT_SBPRO_SYNTH
Sound Blaster Pro FM synthesizer.
MM_MSFT_SBPRO_AUX_LINE
Sound Blaster Pro auxiliary line input.
MM_MSFT_SBPRO_AUX_CD
Sound Blaster Pro auxiliary CD.
MM_MSFT_SBPRO_MIXER
Sound Blaster Pro mixer.
MM_MSFT_MSOPL_SYNTH
Yamaha OPL2/OPL3-compatible FM synthesizer.

- **MOSCOM Corporation (MM_MOSCOM) Product IDs:**
  MM_MOSCOM_VPC2400
  MOSCOM Four Point Voice Processing / Voice Recognition Board.

- **NCR Corporation (MM_NCR) Product IDs:**
  MM_NCR_BA_WAVEIN
  BA waveform input.
  MM_NCR_BA_WAVEOUT
  BA waveform output.
  MM_NCR_BA_SYNTH
  BA synthesizer.
  MM_NCR_BA_AUX
  BA auxiliary.
  MM_NCR_BA_MIXER
  BA mixer.

- **New Media Corporation (MM_NEWMEDIA) Product IDs:**
  MM_NEWMEDIA_WAVJAMMER
  New Media Jammer waveform.

- **Olivetti (MM_OLIVETTI) Product IDs:**
MM_OLIVETTI_WAVEIN
Olivetti waveform input.

MM_OLIVETTI_WAVEOUT
Olivetti waveform output.

MM_OLIVETTI_MIXER
Olivetti mixer.

MM_OLIVETTI_AUX
Olivetti auxiliary.

MM_OLIVETTI_MIDIIN
Olivetti MIDI input.

MM_OLIVETTI_MIDIOUT
Olivetti MIDI output.

MM_OLIVETTI_SYNT
Olivetti synthesizer.

MM_OLIVETTI_JOYSTICK
Olivetti joystick.

MM_OLIVETTI_ACM_GSM
Olivetti ACM GSM.

MM_OLIVETTI_ACM_ADPCM
Olivetto ACM ADPCM.

MM_OLIVETTI_ACM_SBC
Olivetti ACM SBC.

MM_OLIVETTI_ACM_OPR
Olivetti ACM OPR.

- **OPTi Computers, Inc. (MM_OPTI) Product IDs:**

  MM_OPTI_M16_FMSYNTH_STEREO
  M16 FM stereo synthesizer.

  MM_OPTI_M16_MIDIIN
  M16 MIDI input.

  MM_OPTI_M16_MIDIOUT
  M16 MIDI output.

  MM_OPTI_M16_WAVEIN
  M16 waveform input.

  MM_OPTI_M16_WAVEOUT
  M16 waveform output.

  MM_OPTI_M16_MIXER
  M16 mixer.

  MM_OPTI_M16_AUX
  M16 auxiliary.

  MM_OPTI_M32_WAVEIN
  M32 waveform input.

  MM_OPTI_M32_WAVEOUT
  M32 waveform output.
MM_OPTI_M32_MIDIIN
  M32 MIDI input.
MM_OPTI_M32_MIDIOUT
  M32 MIDI output.
MM_OPTI_M32_SYNTH_STEREO
  M32 stereo synthesizer.
MM_OPTI_M32_MIXER
  M32 mixer.
MM_OPTI_M32_AUX
  M32 auxiliary.
MM_OPTI_P16_FMSYNTH_STEREO
  P16 FM stereo synthesizer.
MM_OPTI_P16_MIDIIN
  P16 MIDI input.
MM_OPTI_P16_MIDIOUT
  P16 MIDI output.
MM_OPTI_P16_WAVEIN
  P16 waveform input.
MM_OPTI_P16_WAVEOUT
  P16 waveform output.
MM_OPTI_P16_MIXER
  P16 mixer.
MM_OPTI_P16_AUX
  P16 auxiliary.

- **Roland (MM_ROLAND) Product IDs:**
  MM_ROLAND_MPU401_MIDIOUT
    MPU401 MIDI output.
  MM_ROLAND_MPU401_MIDIIN
    MPU401 MIDI input.
  MM_ROLAND_SC7_MIDIOUT
    SC7 MIDI output.
  MM_ROLAND_SC7_MIDIIN
    SC7 MIDI input.
  MM_ROLAND_SERIAL_MIDIOUT
    Serial MIDI output.
  MM_ROLAND_SERIAL_MIDIIN
    Serial MIDI input.
  MM_ROLAND_SMPU_MIDIOUTA
    SMPU MIDI output A.
  MM_ROLAND_SMPU_MIDIOUTB
    SMPU MIDI output B.
  MM_ROLAND_SMPU_MIDIINA
    SMPU MIDI input A.
MM_ROLAND_SMPU_MIDIINB
  SMPU MIDI input B.

- **Sierra Semiconductor Corp. (MM_SIERRA) Product IDs:**
  MM_SIERRA_ARIA_MIDIOUT
  Aria MIDI output.
  MM_SIERRA_ARIA_MIDIIN
  Aria MIDI input.
  MM_SIERRA_ARIA_SYNT
  Aria synthesizer.
  MM_SIERRA_ARIA_WAVEOUT
  Aria waveform output.
  MM_SIERRA_ARIA_WAVEIN
  Aria waveform input.
  MM_SIERRA_ARIA_AUX
  Aria auxiliary.
  MM_SIERRA_ARIA_AUX2
  Aria auxiliary 2.

- **Silicon Soft, Inc. (MM_SILICONSOFT) Product IDs:**
  MM_SILICONSOFT_SC1_WAVEIN
  High-sample-rate waveform input.
  MM_SILICONSOFT_SC1_WAVEOUT
  High-sample-rate waveform output.
  MM_SILICONSOFT_SC2_WAVEIN
  High-sample-rate, two-channel waveform input.
  MM_SILICONSOFT_SC2_WAVEOUT
  High-sample-rate, two-channel waveform output.
  MM_SILICONSOFT_SOUNDJR2_WAVEOUT
  Self-powered waveform output.
  MM_SILICONSOFT_SOUNDJR2PR_WAVEIN
  Self-powered waveform input.
  MM_SILICONSOFT_SOUNDJR2PR_WAVEOUT
  Self-powered, two-channel waveform output.
  MM_SILICONSOFT_SOUNDJR3_WAVEIN
  Self-powered, two-channel waveform input.

- **Tandy Corporation (MM_TANDY) Product IDs:**
  MM_TANDY_VISWAVEIN
  VIS waveform input.
  MM_TANDY_VISWAVEOUT
  VIS waveform output.
  MM_TANDY_VISBIOSSYNTH
  VIS BIOS synthesizer.
  MM_TANDY_SENS_MMAWAVEIN
  SENS MMA waveform input.
Windows API Guide: Product Identifiers

- **Truevision (MM_TRUEVISION) Product IDs:**
  - MM_TRUEVISION_WAVEIN1
    Truevision waveform input.
  - MM_TRUEVISION_WAVEOUT1
    Truevision waveform output.

- **Videologic (MM_VIDEOLOGIC) Product IDs:**
  - MM_VIDEOLOGIC_MSWAVEIN
    Videologic waveform input.
  - MM_VIDEOLOGIC_MSWAVEOUT
    Videologic waveform output.

- **Vitec Multimedia (MM_VITEC) Product IDs:**
  - MM_VITEC_VMAKER
    Vitec VMaker.
  - MM_VITEC_VMPRO
    Vitec VMaker Pro.

- **Vocaltec Ltd. (MM_VOCALTEC) Product IDs:**
  - MM_VOCALTEC_WAVEOUT
    Vocaltec waveform output.
  - MM_VOCALTEC_WAVEIN
    Vocaltec waveform input.

- **Wang Laboratories, Inc. (MM_WANGLABS) Product IDs:**
  - MM_WANGLABS_WAVEIN1
    Wang Laboratories waveform input.
  - MM_WANGLABS_WAVEOUT1
    Wang Laboratories waveform output.

- **Winnov, Inc. (MM_WINNOV) Product IDs:**
  - MM_WINNOV_CAVIAR_WAVEIN
    Caviar waveform input.
  - MM_WINNOV_CAVIAR_WAVEOUT
    Caviar waveform output.
  - MM_WINNOV_CAVIAR_VIDC
Caviar VIDC.
MM_WINNOV_CAVIAR_CHAMPAGNE
   Caviar Fourcc Champagne.
MM_WINNOV_CAVIAR_YUV8
   Caviar Fourcc YUV8.

- **Yamaha Corporation of America (MM_YAMAHA) Product IDs:**
  
  MM_YAMAHA_GSS_SYNTH
  GSS synthesizer.

  MM_YAMAHA_GSS_WAVEOUT
  GSS waveform output.

  MM_YAMAHA_GSS_WAVEIN
  GSS waveform input.

  MM_YAMAHA_GSS_MIDIOUT
  GSS MIDI output.

  MM_YAMAHA_GSS_MIDIIN
  GSS MIDI input.

  MM_YAMAHA_GSS_AUX
  GSS auxiliary.

### Constant Definitions

```
Const MM_APT_ACE100CD = 1
Const MM_ARTISOFT_SBWAVEIN = 1
Const MM_ARTISOFT_SBWAVEOUT = 2
Const MM_AZTECH_AUX_CD = 401
Const MM_AZTECH_AUX_LINE = 402
Const MM_AZTECH_AUX_MIC = 403
Const MM_AZTECH_AUX = 404
Const MM_AZTECH_DSP16_WAVEIN = 65
Const MM_AZTECH_DSP16_WAVEOUT = 66
Const MM_AZTECH_DSP16_FMSYNTH = 68
Const MM_AZTECH_DSP16_WAVESYNTH = 70
Const MM_AZTECH_NOVA16_WAVEIN = 71
Const MM_AZTECH_NOVA16_WAVEOUT = 72
Const MM_AZTECH_NOVA16_MIXER = 73
Const MM_AZTECH_PRO16_WAVEIN = 33
Const MM_AZTECH_PRO16_WAVEOUT = 34
Const MM_AZTECH_WASH16_WAVEIN = 74
Const MM_AZTECH_WASH16_WAVEOUT = 75
Const MM_AZTECH_WASH16_MIXER = 76
Const MM_AZTECH_MIDIOUT = 3
Const MM_AZTECH_MIDIIN = 4
```
Const MM_AZTECH_WAVEIN = 17
Const MM_AZTECH_WAVEOUT = 18
Const MM_AZTECH_FMSYNTH = 20
Const MM_AZTECH_MIXER = 21
Const MM_CAT_WAVEOUT = 1
Const MM_CREATIVE_AUX_CD = 401
Const MM_CREATIVE_AUX_LINE = 402
Const MM_CREATIVE_AUX_MIC = 403
Const MM_CREATIVE_AUX_MASTER = 404
Const MM_CREATIVE_AUX_PCSPK = 405
Const MM_CREATIVE_AUX_WAVE = 406
Const MM_CREATIVE_AUX_MIDI = 407
Const MM_CREATIVE_SB15_WAVEIN = 1
Const MM_CREATIVE_SB15_WAVEOUT = 101
Const MM_CREATIVE_SB16_MIXER = 409
Const MM_CREATIVE_SB20_WAVEIN = 2
Const MM_CREATIVE_SB20_WAVEOUT = 102
Const MM_CREATIVE_SBP16_WAVEIN = 4
Const MM_CREATIVE_SBP16_WAVEOUT = 104
Const MM_CREATIVE_SBPRO_WAVEIN = 3
Const MM_CREATIVE_SBPRO_WAVEOUT = 103
Const MM_CREATIVE_SBPRO_MIXER = 408
Const MM_CREATIVE_MIDIOUT = 201
Const MM_CREATIVE_MIDIIN = 202
Const MM_CREATIVE_FMSYNTH_MONO = 301
Const MM_CREATIVE_FMSYNTH_STEREO = 302
Const MM_CREATIVE_MIDI_AWE32 = 303
Const MM_DSP_GROUP_TRUESPEECH = &H1
Const MM_DSP_SOLUTIONS_WAVEOUT = 1
Const MM_DSP_SOLUTIONS_WAVEIN = 2
Const MM_DSP_SOLUTIONS_SYNTH = 3
Const MM_DSP_SOLUTIONS_AUX = 4
Const MM_ECHO_SYNTH = &H1
Const MM_ECHO_WAVEOUT = &H2
Const MM_ECHO_WAVEIN = &H3
Const MM_ECHO_MIDIOUT = &H4
Const MM_ECHO_MIDIIN = &H5
Const MM_ECHO_AUX = &H6
Const MM_ESS_AMWAVEOUT = &H1
Const MM_ESS_AMWAVEIN = &H2
Const MM_ESS_AMAUX = &H3
Const MM_ESS_AMSYNTH = &H4
Const MM_ESS_AMMIDIOUT = &H5
Const MM_ESS_AMMIDIIN = &H6
Const MM_ESS_AUX_CD = &H8
Const MM_ESS_ES488_WAVEOUT = &H10
Const MM_ESS_ES488_WAVEIN = &H11
Const MM_ESS_ES488_MIXER = &H12
Const MM_ESS_ES688_WAVEOUT = &H13
Const MM_ESS_ES688_WAVEIN = &H14
Const MM_ESS_ES688_MIXER = &H15
Const MM_ESS_ES1488_WAVEOUT = &H16
Const MM_ESS_ES1488_WAVEIN = &H17
Const MM_ESS_ES1488_MIXER = &H18
Const MM_ESS_ES1688_WAVEOUT = &H19
Const MM_ESS_ES1688_WAVEIN = &H1A
Const MM_ESS_ES1688_MIXER = &H1B
Const MM_ESS_MPU401_MIDIOUT = &H9
Const MM_ESS_MPU401_MIDIIN = &HA
Const MM_ESS_MIXER = &H7
Const MM_EVEREX_CARRIER = &H1
Const MM_IBM_PCMCIA_WAVEIN = 11
Const MM_IBM_PCMCIA_WAVEOUT = 12
Const MM_IBM_PCMCIA_SYNTH = 13
Const MM_IBM_PCMCIA_MIDIIN = 14
Const MM_IBM_PCMCIA_MIDIOUT = 15
Const MM_IBM_PCMCIA_AUX = 16
Const MM_MMOTION_WAVEAUX = 1
Const MM_MMOTION_WAVEOUT = 2
Const MM_MMOTION_WAVEIN = 3
Const MM_ICS_WAVEDECK_WAVEOUT = 1
Const MM_ICS_WAVEDECK_WAVEIN = 2
Const MM_ICS_WAVEDECK_MIXER = 3
Const MM_ICS_WAVEDECK_AUX = 4
Const MM_ICS_WAVEDECK_SYNTH = 5
Const MM_INTERACTIVE_WAVEIN = &H45
Const MM_INTERACTIVE_WAVEOUT = &H45
Const MM_IOMAGIC_TEMPO_WAVEOUT = 1
Const MM_IOMAGIC_TEMPO_WAVEIN = 2
Const MM_IOMAGIC_TEMPO_SYNTH = 3
Const MM_IOMAGIC_TEMPO_MIDIOUT = 4
Const MM_IOMAGIC_TEMPO_MXDOUT = 5
Const MM_IOMAGIC_TEMPO_AUXOUT = 6
Const MM_ITERATED_SYS_FUF_CODEC = 1
Const MM_KORG_PCIF_MIDIOUT = 1
Const MM_KORG_PCIF_MIDIIN = 2
Const MM_LYRRUS_BRIDGE_GUITAR = 1
Const MM_MATSUSHITA_WAVEIN = 1
Const MM_MATSUSHITA_WAVEOUT = 2
Const MM_MATSUSHITA_FMSYNTH_STEREO = 3
Const MM_MATSUSHITA_MIXER = 4
Const MM_MATSUSHITA_AUX = 5
Const MM_CDPC_MIDIOUT = &H71
Const MM_CDPC_MIDIIN = &H72
Const MM_CDPC_SYNTH = &H73
Const MM_CDPC_WAVEOUT = &H74
Const MM_CDPC_WAVEIN = &H75
Const MM_CDPC_MIXER = &H76
Const MM_CDPC_AUX = &H77
Const MM_OPUS1208_MIDIOUT = &H81
Const MM_OPUS1208_MIDIIN = &H82
Const MM_OPUS1208_SYNTH = &H83
Const MM_OPUS1208_WAVEOUT = &H84
Const MM_OPUS1208_WAVEIN = &H85
Const MM_OPUS1208_MIXER = &H86
Const MM_OPUS1208_AUX = &H87
Const MM_OPUS1216_MIDIOUT = &H91
Const MM_OPUS1216_MIDIIN = &H92
Const MM_OPUS1216_SYNTH = &H93
Const MM_OPUS1216_WAVEOUT = &H94
Const MM_OPUS1216_WAVEIN = &H95
Const MM_OPUS1216_MIXER = &H96
Const MM_OPUS1216_AUX = &H97
Const MM_PROAUD_MIDIOUT = &H11
Const MM_PROAUD_MIDIIN = &H12
Const MM_PROAUD_SYNTH = &H13
Const MM_PROAUD_WAVEOUT = &H14
Const MM_PROAUD_WAVEIN = &H15
Const MM_PROAUD_MIXER = &H16
Const MM_PROAUD_AUX = &H17
Const MM_PROAUD_16_MIDIOUT = &H61
Const MM_PROAUD_16_MIDIIN = &H62
Const MM_PROAUD_16_SYNTH = &H63
Const MM_PROAUD_16_WAVEOUT = &H64
Const MM_PROAUD_16_WAVEIN = &H65
Const MM_PROAUD_16_MIXER = &H66
Const MM_PROAUD_16_AUX = &H67
Const MM_PROAUD_PLUS_MIDIOUT = &H51
Const MM_PROAUD_PLUS_MIDIIN = &H52
Const MM_PROAUD_PLUS_SYNTH = &H53
Const MM_PROAUD_PLUS_WAVEOUT = &H54
Const MM_PROAUD_PLUS_WAVEIN = &H55
Const MM_PROAUD_PLUS_MIXER = &H56
Const MM_PROAUD_PLUS_AUX = &H57
Const MM_STUDIO_16_MIDIOUT = &H61
Const MM_STUDIO_16_MIDIIN = &H62
Const MM_STUDIO_16_SYNTH = &H63
Const MM_STUDIO_16_WAVEOUT = &H64
Const MM_STUDIO_16_WAVEIN = &H65
Const MM_STUDIO_16_MIXER = &H66
Const MM_STUDIO_16_AUX = &H67
Const MM_THUNDER_SYNTH = &H23
Const MM_THUNDER_WAVEOUT = &H25
Const MM_THUNDER_WAVEIN = &H24
Const MM_THUNDER_AUX = &H27
Const MM_TPORT_WAVEOUT = &H41
Const MM_TPORT_WAVEIN = &H42
Const MM_TPORT_SYNTH = &H43
Const MM_MELABS_MIDI2GO = &H1
Const MM_METHEUS_ZIPPER = 1
Const MM_MIDI_MAPPER = 1
Const MM_WAVE_MAPPER = 2
Const MM_SNDBLST_MIDIOUT = 3
Const MM_SNDBLST_MIDIIN = 4
Const MM_SNDBLST_SYNTH = 5
Const MM_SNDBLST_WAVEOUT = 6
Const MM_SNDBLST_WAVEIN = 7
Const MM_ADLIB = 9
Const MM_MPU401_MIDIOUT = 10
Const MM_MPU401_MIDIIN = 11
Const MM_PC_JOYSTICK = 12
Const MM_PCSPEAKER_WAVEOUT = 13
Const MM_MSFT_WSS_WAVEIN = 14
Const MM_MSFT_WSS_WAVEOUT = 15
Const MM_MSFT_WSS_FMSYNTH_STEREO = 16
Const MM_MSFT_WSS_MIXER = 17
Const MM_MSFT_WSS_OEM_WAVEIN = 18
Const MM_MSFT_WSS_OEM_WAVEOUT = 19
Const MM_MSFT_WSS_OEM_FMSYNTH_STEREO = 20
Const MM_MSFT_WSS_AUX = 21
Const MM_MSFT_WSS_OEM_AUX = 22
Const MM_MSFT_GENERIC_WAVEIN = 23
Const MM_MSFT_GENERIC_WAVEOUT = 24
Const MM_MSFT_GENERIC_MIDIIN = 25
Const MM_MSFT_GENERIC_MIDIOUT = 26
Const MM_MSFT_GENERIC_MIDISYNTH = 27
Const MM_MSFT_GENERIC_AUX_LINE = 28
Const MM_MSFT_GENERIC_AUX_MIC = 29
Const MM_MSFT_GENERIC_AUX_CD = 30
Const MM_MSFT_WSS_OEM_MIXER = 31
Const MM_MSFT_MSACM = 32
Const MM_MSFT_ACM_MSADPCM = 33
Const MM_MSFT_ACM_IMAADPCM = 34
Const MM_MSFT_ACM_MSFILTER = 35
Const MM_MSFT_ACM_GSM610 = 36
Const MM_MSFT_ACM_G711 = 37
Const MM_MSFT_ACM_PCM = 38
Const MM_WSS_SB16_WAVEIN = 39
Const MM_WSS_SB16_WAVEOUT = 40
Const MM_WSS_SB16_MIDIIN = 41
Const MM_WSS_SB16_MIDIOUT = 42
Const MM_WSS_SB16_SYNTH = 43
Const MM_WSS_SB16_AUX_LINE = 44
Const MM_WSS_SB16_AUX_CD = 45
Const MM_WSS_SB16_MIXER = 46
Const MM_WSS_SBPRO_WAVEIN = 47
Const MM_WSS_SBPRO_WAVEOUT = 48
Const MM_WSS_SBPRO_MIDIIN = 49
Const MM_WSS_SBPRO_MIDIOUT = 50
Const MM_WSS_SBPRO_SYNTH = 51
Const MM_WSS_SBPRO_AUX_LINE = 52
Const MM_WSS_SBPRO_AUX_CD = 53
Const MM_WSS_SBPRO_MIXER = 54
Const MM_MSFT_WSS_NT_WAVEIN = 55
Const MM_MSFT_WSS_NT_WAVEOUT = 56
Const MM_MSFT_WSS_NT_FMSYNTH_STEREO = 57
Const MM_MSFT_WSS_NT_MIXER = 58
Const MM_MSFT_WSS_NT_AUX = 59
Const MM_MSFT_SB16_WAVEIN = 60
Const MM_MSFT_SB16_WAVEOUT = 61
Const MM_MSFT_SB16_MIDIIN = 62
Const MM_MSFT_SB16_MIDIOUT = 63
Const MM_MSFT_SB16_SYNTH = 64
Const MM_MSFT_SB16_AUX_LINE = 65
Const MM_MSFT_SB16_AUX_CD = 66
Const MM_MSFT_SB16_MIXER = 67
Const MM_MSFT_SBPRO_WAVEIN = 68
Const MM_MSFT_SBPRO_WAVEOUT = 69
Const MM_MSFT_SBPRO_MIDIIN = 70
Const MM_MSFT_SBPRO_MIDIOUT = 71
Const MM_MSFT_SBPRO_SYNTH = 72
Const MM_MSFT_SBPRO_AUX_LINE = 73
Const MM_MSFT_SBPRO_AUX_CD = 74
Const MM_MSFT_SBPRO_MIXER = 75
Const MM_MSFT_MSOPL_SYNTH = 76
Const MM_MOSCOM_VPC2400 = 1
Const MM_NCR_BA_WAVEIN = 1
Const MM_NCR_BA_WAVEOUT = 2
Const MM_NCR_BA_SYNTH = 3
Const MM_NCR_BA_AUX = 4
Const MM_NCR_BA_MIXER = 5
Const MM_NEWMEDIA_WAVJAMMER = 1
Const MM_OLIVETTI_WAVEIN = 1
Const MM_OLIVETTI_WAVEOUT = 2
Const MM_OLIVETTI_MIXER = 3
Const MM_OLIVETTI_AUX = 4
Const MM_OLIVETTI_MIDIIN = 5
Const MM_OLIVETTI_MIDIOUT = 6
Const MM_OLIVETTI_SYNTH = 7
Const MM_OLIVETTI_JOYSTICK = 8
Const MM_OLIVETTI_ACM_GSM = 9
Const MM_OLIVETTI_ACM_ADPCM = 10
Const MM_OLIVETTI_ACM_SBC = 12
Const MM_OLIVETTI_ACM_OPR = 13
Const MM_OPTI_M16_FMSYNTH_STEREO = &H1
Const MM_OPTI_M16_MIDIIN = &H2
Const MM_OPTI_M16_MIDIOUT = &H3
Const MM_OPTI_M16_WAVEIN = &H4
Const MM_OPTI_M16_WAVEOUT = &H5
Const MM_OPTI_M16_MIXER = &H6
Const MM_OPTI_M16_AUX = &H7
Const MM_OPTI_M32_WAVEIN = &H20
Const MM_OPTI_M32_WAVEOUT = &H21
Const MM_OPTI_M32_MIDIIN = &H22
Const MM_OPTI_M32_MIDIOUT = &H23
Const MM_OPTI_M32_SYNTH_STEREO = &H24
Const MM_OPTI_M32_MIXER = &H25
Const MM_OPTI_M32_AUX = &H26
Const MM_OPTI_P16_FMSYNTH_STEREO = &H10
Const MM_OPTI_P16_MIDIIN = &H11
Const MM_OPTI_P16_MIDIOUT = &H12
Const MM_OPTI_P16_WAVEIN = &H13
Const MM_OPTI_P16_WAVEOUT = &H14
Const MM_OPTI_P16_MIXER = &H15
Const MM_OPTI_P16_AUX = &H16
Const MM_ROLAND_MPU401_MIDIOUT = 15
Const MM_ROLAND_MPU401_MIDIIN = 16
Const MM_ROLAND_SC7_MIDIOUT = 21
Const MM_ROLAND_SC7_MIDIIN = 22
Const MM_ROLAND_SERIAL_MIDIOUT = 23
Const MM_ROLAND_SERIAL_MIDIIN = 24
Const MM_ROLAND_SMPU_MIDIOUTA = 17
Const MM_ROLAND_SMPU_MIDIOUTB = 18
Const MM_ROLAND_SMPU_MIDIINA = 19
Const MM_ROLAND_SMPU_MIDIINB = 20
Const MM_SIERRA_ARIA_MIDIOUT = &H14
Const MM_SIERRA_ARIA_MIDIIN = &H15
Const MM_SIERRA_ARIA_SYNTH = &H16
Const MM_SIERRA_ARIA_WAVEOUT = &H17
Const MM_SIERRA_ARIA_WAVEIN = &H18
Const MM_SIERRA_ARIA_AUX = &H19
Const MM_SIERRA_ARIA_AUX2 = &H20
Const MM_SILICONSOFT_SC1_WAVEIN = 1
Const MM_SILICONSOFT_SC1_WAVEOUT = 2
Const MM_SILICONSOFT_SC2_WAVEIN = 3
Const MM_SILICONSOFT_SC2_WAVEOUT = 4
Const MM_SILICONSOFT_SOUNDJR2_WAVEOUT = 5
Const MM_SILICONSOFT_SOUNDJR2PR_WAVEIN = 6
Const MM_SILICONSOFT_SOUNDJR2PR_WAVEOUT = 7
Const MM_SILICONSOFT_SOUNDJR3_WAVEIN = 8
Const MM_TANDY_VISWAVEIN = 1
Const MM_TANDY_VISWAVEOUT = 2
Const MM_TANDY_VISBIOSSYNTH = 3
Const MM_TANDY_SENS_MMAWAVEIN = 4
Const MM_TANDY_SENS_MMAWAVEOUT = 5
Const MM_TANDY_SENS_MMAMIDIIN = 6
Const MM_TANDY_SENS_MMAMIDIOUT = 7
Const MM_TANDY_SENS_VISWAVEOUT = 8
Const MM_TANDY_PSSJWAVEIN = 9
Const MM_TANDY_PSSJWAVEOUT = 10
Const MM_TRUEVISION_WAVEIN1 = 1
Const MM_TRUEVISION_WAVEOUT1 = 2
Const MM_VIDEOTOLOGY_MSWAVEIN = 1
Const MM_VIDEOTOLOGY_MSWAVEOUT = 2
Const MM_VITEC_VMAKER = 1
Const MM_VITEC_VMPRO = 2
Const MM_VOCALTEC_WAVEOUT = 1
Const MM_VOCALTEC_WAVEIN = 2
Const MM_WANGLABS_WAVEIN1 = 1
Const MM_WANGLABS_WAVEOUT1 = 2
Const MM_WINNOV_CAVIAR_WAVEIN = 1
Const MM_WINNOV_CAVIAR_WAVEOUT = 2
Const MM_WINNOV_CAVIAR_VIDC = 3
Const MM_WINNOV_CAVIAR_CHAMPAGNE = 4
Const MM_WINNOV_CAVIAR_YUV8 = 5
Const MM_YAMAHA_GSS_SYNTH = &H1
Const MM_YAMAHA_GSS_WAVEOUT = &H2
Const MM_YAMAHA_GSS_WAVEIN = &H3
Const MM_YAMAHA_GSS_MIDIOUT = &H4
Const MM_YAMAHA_GSS_MIDIIN = &H5
Const MM_YAMAHA_GSS_AUX = &H6

Used By

AUXCAPS, JOYCAPS, WAVEOUTCAPS

Go back to the Other Information listing.
Go back to the Reference section index.

Last Modified: September 9, 1999
This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/productids.html
Virtual-Key Codes

Description & Usage

The virtual-key codes identify various virtual keys. Virtual keys mainly consist of actual keyboard keys, but also include "virtual" elements such as the three mouse buttons. The virtual keys also include many "keys" which usually do not exist at all! A key's virtual-key code does not change when modifier keys (Ctrl, Alt, Shift, etc.) are held -- e.g., the 1 key has the same virtual-key code whether 1 or ! is pressed. However, the numbers in the numeric keypad on the keyboard do have two different virtual-key codes: one for when Num Lock is on, and another for when Num Lock is off. Note that the virtual-key codes of 0-9 and A-Z equal their ASCII codes.

Note: The actual meanings of some of the key codes may vary on keyboards designed for different languages. Most notably, the VK_OEM_* that denote punctuation keys may vary between languages, relating to a different punctuation key. The meanings listed below are for a U.S. English-language keyboard.

Virtual Key Codes

VK_LBUTTON
  The left mouse button
VK_RBUTTON
  The right mouse button
VK_CANCEL
  The Cancel virtual key, used for control-break processing
VK_MBUTTON
  The middle mouse button
VK_BACK
  Backspace
VK_TAB
  Tab
VK_CLEAR
  5 (keypad without Num Lock)
VK_RETURN
Enter
VK_SHIFT
    Shift (either one)
VK_CONTROL
    Ctrl (either one)
VK_MENU
    Alt (either one)
VK_PAUSE
    Pause
VK_CAPITAL
    Caps Lock
VK_ESCape
    Esc
VK_SPACE
    Spacebar
VK_PRIOR
    Page Up
VK_NEXT
    Page Down
VK_END
    End
VK_HOME
    Home
VK_LEFT
    Left Arrow
VK_UP
    Up Arrow
VK_RIGHT
    Right Arrow
VK_DOWN
    Down Arrow
VK_SELECT
    Select
VK_PRINT
    Print (only used by Nokia keyboards)
VK_EXECUTE
    Execute (not used)
VK_SNAPSHOT
    Print Screen
VKINSERT
    Insert
VK_DELETE
    Delete
VK_HELP
    Help
VK_0
    0
VK_1
    1
VK_2
    2
VK_3
    3
VK_4
    4
VK_5
    5
VK_6
    6
VK_7
    7
VK_8
    8
VK_9
    9
VK_A
    A
VK_B
    B
VK_C
    C
VK_D
    D
VK_E
    E
VK_F
    F
VK_G
    G
VK_H
    H
VK_I
    I
VK_J
    J
VK_K
Windows API Guide: Virtual-Key Codes

K
VK_L
L
VK_M
M
VK_N
N
VK_O
O
VK_P
P
VK_Q
Q
VK_R
R
VK_S
S
VK_T
T
VK_U
U
VK_V
V
VK_W
W
VK_X
X
VK_Y
Y
VK_Z
Z
VK_STARTKEY
Start Menu key
VK_CONTEXTKEY
Context Menu key
VK_NUMPAD0
0 (keypad with Num Lock)
VK_NUMPAD1
1 (keypad with Num Lock)
VK_NUMPAD2
2 (keypad with Num Lock)
VK_NUMPAD3
3 (keypad with Num Lock)
VK_NUMPAD4
  4 (keypad with Num Lock)
VK_NUMPAD5
  5 (keypad with Num Lock)
VK_NUMPAD6
  6 (keypad with Num Lock)
VK_NUMPAD7
  7 (keypad with Num Lock)
VK_NUMPAD8
  8 (keypad with Num Lock)
VK_NUMPAD9
  9 (keypad with Num Lock)
VK_MULTIPLY
  * (keypad)
VK_ADD
  + (keypad)
VK_SEPARATOR
  Separator (never generated by the keyboard)
VK_DECIMAL
  . (keypad with Num Lock)
VK_DIVIDE
  / (keypad)
VK_F1
  F1
VK_F2
  F2
VK_F3
  F3
VK_F4
  F4
VK_F5
  F5
VK_F6
  F6
VK_F7
  F7
VK_F8
  F8
VK_F9
  F9
VK_F10
  F10
VK_F11
F11
VK_F12
  F12
VK_F13
  F13
VK_F14
  F14
VK_F15
  F15
VK_F16
  F16
VK_F17
  F17
VK_F18
  F18
VK_F19
  F19
VK_F20
  F20
VK_F21
  F21
VK_F22
  F22
VK_F23
  F23
VK_F24
  F24
VK_NUMLOCK
  Num Lock
VK_OEM_SCROLL
  Scroll Lock
VK_OEM_1
  ;
VK_OEM_PLUS
  =
VK_OEM_COMMA
  ,
VK_OEM_MINUS
  -
VK_OEM_PERIOD
  .
VK_OEM_2
  /

VK_OEM_3
\`
VK_OEM_4
[
VK_OEM_5
\`
VK_OEM_6
]
VK_OEM_7
,
VK_OEM_8
(unknown)
VK_ICO_F17
F17 on Olivetti extended keyboard (internal use only)
VK_ICO_F18
F18 on Olivetti extended keyboard (internal use only)
VK_OEM_102
< or | on IBM-compatible 102 enhanced non-U.S. keyboard
VK_ICO_HELP
Help on Olivetti extended keyboard (internal use only)
VK_ICO_00
00 on Olivetti extended keyboard (internal use only)
VK_ICO_CLEAR
Clear on Olivette extended keyboard (internal use only)
VK_OEM_RESET
Reset (Nokia keyboards only)
VK_OEM_JUMP
Jump (Nokia keyboards only)
VK_OEM_PA1
PA1 (Nokia keyboards only)
VK_OEM_PA2
PA2 (Nokia keyboards only)
VK_OEM_PA3
PA3 (Nokia keyboards only)
VK_OEM_WSCTRL
WSCTRL (Nokia keyboards only)
VK_OEM_CUSEL
CUSEL (Nokia keyboards only)
VK_OEM_ATTN
ATTN (Nokia keyboards only)
VK_OEM_FINNISH
FINNISH (Nokia keyboards only)
VK_OEM_COPY
COPY (Nokia keyboards only)

VK_OEM_AUTO
  AUTO (Nokia keyboards only)

VK_OEM_ENLW
  ENLW (Nokia keyboards only)

VK_OEM_BACKTAB
  BACKTAB (Nokia keyboards only)

VK_ATTN
  ATTN

VK_CRSEL
  CRSEL

VK_EXSEL
  EXSEL

VK_EREOF
  EREOF

VK_PLAY
  PLAY

VK_ZOOM
  ZOOM

VK_NONAME
  NONAME

VK_PA1
  PA1

VK_OEM_CLEAR
  CLEAR

Constant Definitions

Const VK_LBUTTON = &H1
Const VK_RBUTTON = &H2
Const VK_CANCEL = &H3
Const VK_MBUTTON = &H4
Const VK_BACK = &H8
Const VK_TAB = &H9
Const VK_CLEAR = &HC
Const VK_RETURN = &HD
Const VK_SHIFT = &H10
Const VK_CONTROL = &H11
Const VK_MENU = &H12
Const VK_PAUSE = &H13
Const VK_CAPITAL = &H14
Const VK_ESCAPE = &H1B
Const VK_SPACE = &H20
Const VK_PRIOR = &H21
Const VK_NEXT = &H22
Const VK_END = &H23
Const VK_HOME = &H24
Const VK_LEFT = &H25
Const VK_UP = &H26
Const VK_RIGHT = &H27
Const VK_DOWN = &H28
Const VK_SELECT = &H29
Const VK_PRINT = &H2A
Const VK_EXECUTE = &H2B
Const VK_SNAPSHOT = &H2C
Const VK_INSERT = &H2D
Const VK_DELETE = &H2E
Const VK_HELP = &H2F
Const VK_0 = &H30
Const VK_1 = &H31
Const VK_2 = &H32
Const VK_3 = &H33
Const VK_4 = &H34
Const VK_5 = &H35
Const VK_6 = &H36
Const VK_7 = &H37
Const VK_8 = &H38
Const VK_9 = &H39
Const VK_A = &H41
Const VK_B = &H42
Const VK_C = &H43
Const VK_D = &H44
Const VK_E = &H45
Const VK_F = &H46
Const VK_G = &H47
Const VK_H = &H48
Const VK_I = &H49
Const VK_J = &H4A
Const VK_K = &H4B
Const VK_L = &H4C
Const VK_M = &H4D
Const VK_N = &H4E
Const VK_O = &H4F
Const VK_P = &H50
Const VK_Q = &H51
Const VK_R = &H52
Const VK_S = &H53
Const VK_T = &H54
Const VK_U = &H55
Const VK_V = &H56
Const VK_W = &H57
Const VK_X = &H58
Const VK_Y = &H59
Const VK_Z = &H5A
Const VK_STARTKEY = &H5B
Const VK_CONTEXTKEY = &H5D
Const VK_NUMPAD0 = &H60
Const VK_NUMPAD1 = &H61
Const VK_NUMPAD2 = &H62
Const VK_NUMPAD3 = &H63
Const VK_NUMPAD4 = &H64
Const VK_NUMPAD5 = &H65
Const VK_NUMPAD6 = &H66
Const VK_NUMPAD7 = &H67
Const VK_NUMPAD8 = &H68
Const VK_NUMPAD9 = &H69
Const VK_MULTIPLY = &H6A
Const VK_ADD = &H6B
Const VK_SEPARATOR = &H6C
Const VK_SUBTRACT = &H6D
Const VK_DECIMAL = &H6E
Const VK_DIVIDE = &H6F
Const VK_F1 = &H70
Const VK_F2 = &H71
Const VK_F3 = &H72
Const VK_F4 = &H73
Const VK_F5 = &H74
Const VK_F6 = &H75
Const VK_F7 = &H76
Const VK_F8 = &H77
Const VK_F9 = &H78
Const VK_F10 = &H79
Const VK_F11 = &H7A
Const VK_F12 = &H7B
Const VK_F13 = &H7C
Const VK_F14 = &H7D
Const VK_F15 = &H7E
Const VK_F16 = &H7F
Const VK_F17 = &H80
Const VK_F18 = &H81
Const VK_F19 = &H82
Const VK_F20 = &H83
Const VK_F21 = &H84
Const VK_F22 = &H85
Const VK_F23 = &H86
Const VK_F24 = &H87
Const VK_NUMLOCK = &H90
Const VK_OEM_SCROLL = &H91
Const VK_OEM_1 = &HBA
Const VK_OEM_PLUS = &HBB
Const VK_OEM_COMMA = &HBC
Const VK_OEM_MINUS = &HBD
Const VK_OEM_PERIOD = &HBE
Const VK_OEM_2 = &HBF
Const VK_OEM_3 = &HC0
Const VK_OEM_4 = &HDB
Const VK_OEM_5 = &HDC
Const VK_OEM_6 = &HDD
Const VK_OEM_7 = &HDE
Const VK_OEM_8 = &HDF
Const VK_ICO_F17 = &HE0
Const VK_ICO_F18 = &HE1
Const VK_OEM102 = &HE2
Const VK_ICO_HELP = &HE3
Const VK_ICO_00 = &HE4
Const VK_ICO_CLEAR = &HE6
Const VK_OEM_RESET = &HE9
Const VK_OEM_JUMP = &HEA
Const VK_OEM_PA1 = &HEB
Const VK_OEM_PA2 = &HEC
Const VK_OEM_PA3 = &HED
Const VK_OEM_WSCTRL = &HEE
Const VK_OEM_CUSEL = &HEF
Const VK_OEM_ATTN = &HF0
Const VK_OEM_FINNISH = &HF1
Const VK_OEM_COPY = &HF2
Const VK_OEM_AUTO = &HF3
Const VK_OEM_ENLW = &HF4
Const VK_OEM_BACKTAB = &HF5
Const VK_ATTN = &HF6
Const VK_CRSEL = &HF7
Const VK_EXSEL = &HF8
Const VK_EREOF = &HF9
Const VK_PLAY = &HFA
Const VK_ZOOM = &HFB
Const VK_NONAME = &HFC
Const VK_PA1 = &HFD
Const VK_OEM_CLEAR = &HFE

**Used By**

GetAsyncKeyState, GetKeyboardState, GetKeyState, keybd_event, KEYBDINPUT, SetKeyboardState

Go back to the Other Information listing.
Go back to the Reference section index.

---

**Last Modified:** September 6, 1999

This page is copyright © 1999 Paul Kuliniewicz. Copyright Information Revised October 29, 2000

Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail

This page is at http://www.vbapi.com/ref/other/virtualkeycodes.html
Window Classes

Window classes are sets of attributes shared by a group of windows. When a new window is created (for example, by `CreateWindowEx`), it inherits many of its properties from its class. One major use of classes is the creation of controls. Predefined window classes are essentially what makes controls such as buttons or edit boxes appear different from "normal" windows (although all windows belong to one class or another).

Of course, you are not limited to using only the predefined classes. By calling `RegisterClassEx` your program can create its own window classes. However, for most purposes, using one of the predefined classes is best when creating some sort of control, especially if one already exists.

Typically, a program uses these classes to create controls. Each class also has a unique set of window style flags that only apply to it. These unique styles typically control properties specific to that type of control. Below is a list of some of the window classes predefined in the Windows API. Note that this list does not (yet) include all classes defined in the Windows API. Information that applies to all window classes in general appears first in the list.

- **Styles Common to All Classes**
  - Window styles shared by all classes NEW
  - Extended window styles shared by all classes NEW

- **Class-Specific Information**
  - Button control (class "BUTTON") NEW
  - Combo Box control (class "COMBOBOX") NEW
  - Edit control (class "EDIT") NEW
  - IP Address control (class "SysIPAddress32") NEW
  - List Box control (class "LISTBOX") NEW
  - Scroll Bar control (class "SCROLLBAR") NEW
  - Static control (class "STATIC") NEW

Base Window Styles

The following window styles are shared by all windows, regardless of their class. They generally describe the window's general appearance, although many of the styles apply best to non-control windows (particularly overlapped windows).

Styles

**WS_BORDER**
The window has a thin-line border.

**WS_CAPTION**
The window has a title bar.

**WS_CHILD, WS_CHILDWINDOW**
The window is a child window. These windows cannot have menu bars nor can have the WS_POPUP style.

**WS_CLIPCHILDREN**
For a parent window, exclude the areas occupied by child windows when drawing within the window (to avoid drawing on any child windows).

**WS_CLIPSIBLINGS**
For a child window, exclude the areas occupied by fellow children of the window's parent when drawing within the window (to avoid drawing on any sibling windows).

**WS_DISABLED**
The window is disabled.

**WS_DLGFRAME**
The window has a border style typical of dialog boxes. These windows cannot have a title bar.

**WS_GROUP**
Identifies the first control in a group of controls. Any controls following this one are assumed to be part of this control's group, until a control with the WS_GROUP style is encountered.

**WS_HSCROLL**
The window has a horizontal scroll bar.

**WS_MAXIMIZE**
The window is maximized.

**WS_MAXIMIZEBOX**
The window has a maximize button. This cannot be used on windows having the WS_EX_CONTEXTHELP extended window style. The WS_SYSMENU window style must also...
Windows API Guide: Base Window Styles

be specified.

**WS_MINIMIZE, WS_ICONIC**

The window is minimized.

**WS_MINIMIZEBOX**

The window has a minimize button. This cannot be used on windows having the
**WS_EX_CONTEXTHELP** extended window style. The **WS_SYSMENU** window style must also
be specified.

**WS_OVERLAPPED, WS_TILED**

The window is an overlapped window, which as a title bar and a border.

**WS_OVERLAPPEDWINDOW, WS_TILEDWINDOW**

The window is an overlapped window, having a title bar, a sizing border, a title bar, a system
menu, and minimize and maximize boxes.

**WS_POPUP**

The window is a popup window. This cannot be used with the **WS_CHILD** window style.

**WS_POPUPWINDOW**

The window is a popup window, having a thin-line border and system menu. The window can
only be visible if it also has the **WS_CAPTION** window style.

**WS_SIZEBOX**

Same as **WS_THICKFRAME**.

**WS_SYSMENU**

The window has a system menu on its title bar. The **WS_CAPTION** window style must also be
specified.

**WS_TABSTOP**

The control can press Tab repeatedly to set the focus to this control.

**WS_THICKFRAME**

The window has a sizing border.

**WS_VISIBLE**

The window is visible.

**WS_VSCROLL**

The window has a vertical scroll bar.

---

**Constant Definitions**

Const WS_BORDER = &H800000
Const WS_CAPTION = &HC00000
Const WS_CHILD = &H40000000
Const WS_CHILDWINDOW = &H40000000
Const WS_CLIPCHILDREN = &H2000000
Const WS_CLIPSIBLINGS = &H4000000
Const WS_DISABLED = &H8000000
Const WS_DLGFRAME = &H8000000
Const WS_GROUP = &H200000

---
Const WS_HSCROLL = &H100000
Const WS_ICONIC = &H20000000
Const WS_MAXIMIZE = &H100000
Const WS_MAXIMIZEBOX = &H10000
Const WS_MINIMIZE = &H20000000
Const WS_MINIMIZEBOX = &H20000
Const WS_OVERLAPPED = &H0
Const WS_OVERLAPPEDWINDOW = &HCF0000
Const WS_POPUP = &H80000000
Const WS_POPUPWINDOW = &H80880000
Const WS_SIZEBOX = &H40000
Const WS_SYSMENU = &H80000
Const WS_TABSTOP = &H10000
Const WS_THICKFRAME = &H40000
Const WS_TILED = &H0
Const WS_TILEDWINDOW = &HCF0000
Const WS_VISIBLE = &H10000000
Const WS_VSCROLL = &H200000

Back to the Window Classes list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/classes/basestyles.html
Extended Window Styles

Below is a list of the extended window styles. The main difference between these and regular window styles is that these are specified as a separate value. (For example, a call to `CreateWindowEx` takes one parameter for extended window styles and a second parameter for "regular" window styles.) These styles can be applied to any type of window, although many of them apply best to overlapped windows.

Extended Styles

- **WS_EX_ACCEPTFILES**
  The window accepts files via a drag-and-drop operation.

- **WS_EX_APPWINDOW**
  The window also appears on the taskbar whenever it is visible.

- **WS_EX_CLIENTEDGE**
  The window has a border with a sunken edge.

- **WS_EX_CONTEXTHELP**
  A context-help button appears on the title bar. This cannot be used with the `WS_MAXIMIZEBOX` or `WS_MINIMIZEBOX` regular window styles.

- **WS_EX_CONTROLPARENT**
  The window contains children which can be TABbed through.

- **WS_EX_DLGMODALFRAME**
  The window has a modal dialog frame (a double border).

- **WS_EX_LAYERED**
  **Windows 2000:** The window is a layered window.

- **WS_EX_LAYOUTRTL**
  **Windows 2000:** The window's coordinate system places the horizontal origin on the right side, with increasing x values to the left and decreasing x values to the right.

- **WS_EX_LEFT**
  The window has generic left-aligned properties.

- **WS_EX_LEFTSCROLLBAR**
  If the language supports reading order alignment, position the vertical scroll bar (if any) to the left of the client area.

- **WS_EX_LTRREADING**
  Display window text using left-to-right reading.

- **WS_EX_MDICHILD**
The window is an MDI child window.

**WS_EX_NOACTIVATE**

**Windows 2000:** The window is never brought to the foreground as a result of direct user action.

**WS_EX_NOINHERITLAYOUT**

**Windows 2000:** Do not pass the window's layout to its child windows.

**WS_EX_NOPARENTNOTIFY**

If the window is a child window, do not notify the parent window when the child window is created or destroyed.

**WS_EX_OVERLAPPEDWINDOW**

The window has the border of a typical overlapped window.

**WS_EX_PALETTEWINDOW**

The window is a topmost toolbar window with a raised edge, normally used for a floating palette.

**WS_EX_RIGHT**

The window has generic right-aligned properties, if the language supports reading order alignment.

**WS_EX_RIGHTSCROLLBAR**

Display the vertical scroll bar (if any) to the right of the client area.

**WS_EX_RTLREADING**

Display window text using right-to-left reading, if the language supports reading order alignment.

**WS_EX_STATICEDEGE**

The window has a three-dimensional border intended for items which do not accept user input.

**WS_EX_TOOLWINDOW**

The window is designed to be a floating toolbar window, having a small title bar area.

**WS_EX_TOPMOST**

The window appears above all non-topmost windows, even if it is not active.

**WS_EX_TRANSPARENT**

The window appears transparent because its sibling windows below it are drawn first.

**WS_EX_WINDOWEDGE**

The window has a border with a raised edge.

### Constant Definitions

Note: Some of the values of the extended window style flags are not listed. If you know their values, please [e-mail me about them](mailto:).  

```vbnet
Const WS_EX_ACCEPTFILES = &H10
Const WS_EX_APPWINDOW = &H40000
Const WS_EX_CLIENTEDGE = &H200
Const WS_EX_CONTEXTHELP = &H400
Const WS_EX_CONTROLPARENT = &H10000
Const WS_EX_DLGMODALFRAME = &H1
' Const WS_EX_LAYERED = ???
```
' Const WS_EX_LAYOUTRTL = ???
Const WS_EX_LEFT = &H0
Const WS_EX_LEFTSCROLLBAR = &H4000
Const WS_EX_LTRREADING = &H0
Const WS_EX_MDICHILD = &H40
Const WS_EX_NOACTIVATE = &H8000000
' Const WS_EX_NOINHERITLAYOUT = ???
Const WS_EX_NOPARENTNOTIFY = &H4
Const WS_EX_OVERLAPPEDWINDOW = &H300
Const WS_EX_PALETTEWINDOW = &H188
Const WS_EX_RIGHT = &H1000
Const WS_EX_RIGHTSCROLLBAR = &H0
Const WS_EX_RTLREADING = &H2000
Const WS_EX_STATICEDGE = &H20000
Const WS_EX_TOOLWINDOW = &H80
Const WS_EX_TOPMOST = &H8
Const WS_EX_TRANSPARENT = &H20
Const WS_EX_WINDOWEDGE = &H100
Button Control Window Class

The button control's window class describes any type of button control. This includes command buttons, check boxes, and radio boxes. The button class style assigned to a button control determines what type of button it is.

The button control's window class is registered automatically when Windows starts. The name of the class is "BUTTON".

Styles

BS_3STATE
   The button is a check box which has three states: checked, grayed, and cleared.

BS_AUTO3STATE
   The button is a check box which has three states: checked, grayed, and cleared. The button automatically changes its state when the user selects it.

BS_AUTOCHECKBOX
   The button is a check box whose state toggles when the user selects it.

BS_AUTORADIOBUTTON
   The button is a radio button whose state automatically changes to selected (and the state of all other radio buttons in the group to unselected) when the user selects it.

BS_BITMAP
   The button displays a bitmap.

BS_BOTTOM
   The button's text appears at the bottom of the button rectangle.

BS_CENTER
   The button's text appears centered horizontally within the button rectangle.

BS_CHECKBOX
   The button is a check box.

BS_DEFPUSHBUTTON
   The button is the default push button in a dialog box, having a heavy black border.

BS_GROUPBOX
   The window is a rectangular grouping frame in which other controls can be grouped.

BS_FLAT
   The button is flat, not using the default 3D shading.
BS_ICON
   The button displays an icon.

BS_LEFT
   The text in the button rectangle is left-justified.

BS_LEFTTEXT, BS_RIGHTBUTTON
   The radio or check box's text appears to the left of the button instead of to the right.

BS_MULTILINE
   The button's text is wrapped across multiple lines if it cannot fit on a single line.

BS_NOTIFY
   The button sends focus notification messages to its parent window.

BS_OWNERDRAW
   The owner of the window is responsible for manually drawing it whenever it needs to be redrawn. This window style cannot be combined with any other button styles.

BS_PUSHBUTTON
   The button is a push button.

BS_PUSHLIKE
   The radio or check button takes on the appearance of a push button. A checked state makes the button look depressed; a cleared state makes the button look normal.

BS_RADIOBUTTON
   The button is a radio button.

BS_RIGHT
   The text in the button rectangle is right-justified.

BS_TEXT
   The button displays text.

BS_TOP
   The text appears at the top of the button rectangle.

BS_USERBUTTON
   Obsolete; use BS_OWNERDRAW instead.

BS_VCENTER
   The text appears centered vertically within the button rectangle.

Constant Definitions

Const BS_3STATE = &H5
Const BS_AUTO3STATE = &H6
Const BS_AUTOCHECKBOX = &H3
Const BS_AUTORADIOBOX = &H9
Const BS_BITMAP = &H80
Const BS_BOTTOM = &H800
Const BS_CENTER = &H300
Const BS_CHECKBOX = &H2
Const BS_DEFPUSHBUTTON = &H1
Const BS_FLAT = &H8000
Const BS_GROUPBOX = &H7
Const BS_ICON = &H40
Const BS_LEFT = &H100
Const BS_LEFTTEXT = &H20
Const BS_MULTILINE = &H2000
Const BS_NOTIFY = &H4000
Const BS_OWNERDRAW = &HB
Const BS_PUSHBUTTON = &H0
Const BS_PUSHLIKE = &H1000
Const BS_RADIOBUTTON = &H4
Const BS_RIGHT = &H200
Const BS_RIGHTBUTTON = &H20
Const BS_TEXT = &H0
Const BS_TOP = &H400
Const BS_USERBUTTON = &H8
Const BS_VCENTER = &HC00
Combo Box Control Window Class

The combo box control's window class describes an ordinary combo box control.

The button control's window class is registered automatically when Windows starts. The name of the class is "COMBOBOX".

Styles

CBS_AUTOHSCROLL
Automatically scroll text to the right when the user types a character at the end of the line.

CBS_DISABLENOSCROLL
Show a disable the vertical scroll bar in the drop-down list box if it does not contain enough items to scroll. If this flag is not specified, such a scroll bar is not displayed.

CBS_DROPDOWN
Display the list box whenever the user clicks the drop-down button.

CBS_DROPDOWNLIST
Display the list box whenever the user click the drop-down button, and do not allow the user to change the combo box's selection if the list is not dropped down.

CBS_HASSTRINGS
The combo box is drawn manually by the application and contains strings. Either CBS_OWNERDRAWFIXED or CBS_OWNERDRAWVARIABLE must also be specified.

CBS_LOWERCASE
Convert all text in the combo box to lowercase letters.

CBS_NOINTEGRALHEIGHT
Force the combo box to be exactly the size specified by the application, instead of allowing the operating system to slightly resize it to prevent displaying partial selections.

CBS_OEMCONVERT
Ensure that text in the combo box can readily be converted into the OEM character set.

CBS_OWNERDRAWFIXED
The owner of the combo box is fully responsible for drawing it manually, and all of the items in the combo box have the same height.

CBS_OWNERDRAWVARIABLE
The owner of the combo box is full responsible for drawing it manually, and the heights of the items in the combo box have can be different.
CBS_SIMPLE
   Display the list box portion of the combo box at all times.

CBS_SORT
   Automatically sort the strings added to the combo box.

CBS_UPPERCASE
   Convert all text in the combo box to uppercase letters.

Constant Definitions

Const CBS_AUTOHSCROLL = &H40
Const CBS_DISABLENOSCROLL = &H800
Const CBS_DROPDOWN = &H2
Const CBS_DROPDOWNLIST = &H3
Const CBS_HASSTRINGS = &H200
Const CBS_LOWERCASE = &H4000
Const CBS_NOINTEGRALHEIGHT = &H400
Const CBS_OEMCONVERT = &H80
Const CBS_OWNERDRAWFIXED = &H10
Const CBS_OWNERDRAWVARIABLE = &H20
Const CBS_SIMPLE = &H1
Const CBS_SORT = &H100
Const CBS_UPPERCASE = &H2000
Edit Control Window Class

The edit control's window class describes a regular text edit box.

The button control's window class is registered automatically when Windows starts. The name of the class is "EDIT".

**Styles**

ES_AUTOHSCROLL
Automatically scroll the text to the right when the user types a character at the end of the line. When the user pressed ENTER, scroll the text all the way back to the left.

ES_AUTOVSCROLL
Automatically scroll the text back up when the user presses ENTER on the last line.

ES_CENTER
Center the text horizontally.

ES_LEFT
Left-align the text.

ES_LOWERCASE
Convert all the characters to lowercase as they are typed.

ES_MULTILINE
The edit control displays multiple lines of text. If this flag is not specified, it can only display a single line of text.

ES_NOHIDESEL
Do not hide the selected text in the edit control even if the control loses focus.

ES_NUMBER
Only allow digits to be entered into the edit control.

ES_OEMCONVERT
Ensure that text in the edit control can readily be converted into the OEM character set.

ES_PASSWORD
Display an asterisk for each character typed into the edit control. This cannot be used with ES_MULTILINE.

ES_READONLY
Do not allow the user to edit the text in the control.

ES_RIGHT
Right-align the text.

**ES_UPPERCASE**

Convert all the characters to uppercase as they are typed.

**ES_WANTRETURN**

Insert a carriage return into a multi-line edit control when the user pressed ENTER, instead of implementing the default behavior of ENTER.

### Constant Definitions

```
Const ES_AUTOHSCROLL = &H80  
Const ES_AUTOVSCROLL = &H40  
Const ES_CENTER = &H1       
Const ES_LEFT = &H0        
Const ES_LOWERCASE = &H10   
Const ES_MULTILINE = &H4    
Const ES_NOHIDESEL = &H100  
Const ES_NUMBER = &H2000    
Const ES_OEMCONVERT = &H400 
Const ES_PASSWORD = &H20    
Const ES_READONLY = &H800   
Const ES_RIGHT = &H2       
Const ES_UPPERCASE = &H8    
Const ES_WANTRETURN = &H1000
```

[Back to the Window Class list.](http://www.vbapi.com/ref/other/classes/edit.html)
IP Address Control's Window Class

The **IP Address control**'s window class describes an IP Address control. This control is similar to an edit control, but it is specialized for entering IP addresses in "dotted quad" (a.b.c.d) format. The control performs bounds checking on all inputs, preventing the user from entering something that is not an IP address.

Before creating windows from this class, your program must first register it by calling `InitCommonControlsEx` with the appropriate parameters. The name of the class is "SysIPAddress32".

**Styles**

None.

Back to the Window Class list.
Back to the Reference section.

Last Modified: October 29, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/ref/other/classes/ipaddress.html
List Box Control Window Class

The list box control's window class describes an ordinary list box.

The button control's window class is registered automatically when Windows starts. The name of the class is "LISTBOX".

Styles

LBS_DISABLENOSCROLL
Show a disable the vertical scroll bar in the list box if it does not contain enough items to scroll. If this flag is not specified, such a scroll bar is not displayed.

LBS_EXTENDEDSEL
Allow the user to select multiple items.

LBS_HASSTRINGS
The list box is drawn manually by the application and contains strings. Either LBS_OWNERDRAWFIXED or LBS_OWNERDRAWVARIABLE must also be specified.

LBS_MULTICOLUMN
The list box contains multiple columns which are scrolled horizontally.

LBS_MULTIPLESEL
Toggle the selection of a string every time the user clicks or double-clicks it.

LBS_NODATA
Intended for list boxes having more than 1000 items, do not have the operating system handle any of the entries. LBS_OWNERDRAWFIXED must also be specified, and neither LBS_SORT nor LBS_HASSTRINGS can be specified.

LBS_NOINTEGRALHEIGHT
Force the list box to be exactly the size specified by the application, instead of allowing the operating system to slightly resize it to prevent displaying partial selections.

LBS_NOREDRAW
Do not redraw the list box when changes are made.

LBS_NOSEL
Do not allow the user to select items in the list box.

LBS_NOTIFY
Notify the parent window whenever the user clicks or double-clicks a string in the list box.

LBS_OWNERDRAWFIXED
The owner of the list box is fully responsible for drawing it manually, and all of the items in the list box have the same height.

**LBS_OWNERDRAWVARIABLE**

The owner of the list box is fully responsible for drawing it manually, and the heights of the items in the list box have can be different.

**LBS_SORT**

Automatically sort the strings in the list box alphabetically.

**LBS_STANDARD**

Automatically sort the strings in the list box alphabetically, notify the parent window whenever the user clicks or double-clicks a string, and display a border around the list box.

**LBS_USETABSTOPS**

Allow the list box to expand tab characters when drawing its strings.

**LBS_WANTKEYBOARDINPUT**

Notify the owner of the list box whenever the user types a key while the list box has the focus.

### Constant Definitions

```vba
Const LBS_DISABLENOSCROLL = &H1000
Const LBS_EXTENDEDSEL = &H800
Const LBS_HASSTRINGS = &H40
Const LBS_MULTICOLUMN = &H200
Const LBS_MULTIPLESEL = &H8
Const LBS_NODATA = &H2000
Const LBS_NOINTEGRALHEIGHT = &H100
Const LBS_NOREDRAW = &H4
Const LBS_NOSEL = &H4000
Const LBS_NOTIFY = &H1
Const LBS_OWNERDRAWFIXED = &H10
Const LBS_OWNERDRAWVARIABLE = &H20
Const LBS_SORT = &H2
Const LBS_STANDARD = &HA00006
Const LBS_USETABSTOPS = &H80
Const LBS_WANTKEYBOARDINPUT = &H400
```

[Back to the Window Class list.](#)
[Back to the Reference section.](#)

---

**Last Modified:** October 29, 2000

This page is copyright © 2000 Paul Kuliniewicz. [Copyright Information](#) Revised October 29, 2000
Scroll Bar Control Window Class

The scroll bar control's window class describes a standalone scroll bar (i.e., one that is not part of another control such as an edit control).

The button control's window class is registered automatically when Windows starts. The name of the class is "SCROLLBAR".

Styles

SBS_BOTTOMALIGN
   Align the bottom edge of the scroll bar with the bottom edge of its rectangle, using its default height. SBS_HORZ must also be specified.

SBS_HORZ
   The scroll bar is a horizontal scroll bar.

SBS_LEFTALIGN
   Align the left edge of the scroll bar with the left edge of its rectangle, using its default width. SBS_VERT must also be specified.

SBS_RIGHTALIGN
   Align the right edge of the scroll bar with the right edge of its rectangle, using its default width. SBS_VERT must also be specified.

SBS_SIZEBOX
   The scroll bar is a size box.

SBS_SIZEBOXBOTTOMRIGHTALIGN
   Align the lower-right corner of the size box with the lower-right corner of its rectangle, using its default width and height. SBS_SIZEBOX must also be specified.

SBS_SIZEBOXTOPLEFTALIGN
   Align the upper-left corner of the size box with the upper-left corner of its rectangle, using its default width and height. SBS_SIZEBOX must also be specified.

SBS_SIZEGRIP
   The scroll bar is a size box with a raised edge.

SBS_TOPALIGN
   Align the top edge of the scroll bar with the top edge of its rectangle, using its default height. SBS_HORZ must also be specified.

SBS_VERT
The scroll bar is a vertical scroll bar.

Constant Definitions

Const SBS_BOTTOMALIGN = &H4
Const SBS_HORZ = &H0
Const SBS_LEFTALIGN = &H2
Const SBS_RIGHTALIGN = &H4
Const SBS_SIZEBOX = &H8
Const SBS_SIZEBOXBOTTOMRIGHTALIGN = &H4
Const SBS_SIZEBOXTOPLEFTALIGN = &H2
Const SBS_SIZEGRIP = &H10
Const SBS_TOPALIGN = &H2
Const SBS_VERT = &H1
Static Control Window Class

The static control's window class describes a static control. Static controls are commonly used as labels for other controls.

The button control's window class is registered automatically when Windows starts. The name of the class is "STATIC".

Styles

SS_BITMAP
Display the bitmap specified by the static control's text.

SS_BLACKFRAME
Draw a frame around the static control in the same color as a window frame.

SS_BLACKRECT
Fill the static control with the same color as a window frame.

SS_CENTER
Center the text in the static control.

SS_CENTERIMAGE
If the bitmap or icon is smaller than the size of the static control, fill the rest of the control with whatever color is at the image's upper-left corner.

SS_ENDELLIPSIS
Windows NT, 2000: Replace the end of the string with an ellipsis if it is too long to fit in the static control.

SS_ENHMETAFILE
Display the enhanced metafile identified by the static control's text. Scale the enhanced metafile to fit the static control.

SS_ETCHEDFRAME
Draw the frame of the static control using the etched edge style.

SS_ETCHEDHORZ
Draw only the top and bottom edges of the static control using the etched edge style.

SS_ETCHEDVERT
Draw only the left and right edges of the static control using the etched edge style.

SS_GRAYFRAME
Draw a frame around the static control in the same color as the screen background.
SS_GRAYRECT
Fill the static control with the same color as the screen background.

SS_ICON
Display the icon identified by the static control's text. The static control automatically resizes to the size of the icon.

SS_LEFT
Left-align the text in the static control.

SS_LEFTNOWORDWRAP
Left-align the text in the static control, but do not word wrap.

SS_NOPREFIX
Do not use an amperstand character in the string to identify an accelerator prefix, instead displaying the amperstands as regular characters.

SS_OWNERDRAW
The owner of the static control is fully responsible for drawing the control.

SS_PATHHELLIPSIS
Windows NT, 2000: Replace characters in the middle of a string holding a path with an ellipsis if it is too long to fit in the static control.

SS_REALSIZEIMAGE
Clip an image or bitmap if it does not fit inside the static control instead of resizing the control.

SS_RIGHT
Right-align the text in the static control.

SS_RIGHTJUST
Do not move the lower-right corner of the static control when resizing it to accomodate a bitmap or icon.

SS_SIMPLE
Draw a simple rectangle and display a single line of left-aligned text in the static control.

SS_SUNKEN
Draw a half-sunken border around the static control.

SS_WHITELINE
Draw a frame around the static control in the same color as the window background.

SS_WHITERECT
Fill the static control with the same color as the window background.

SS_WORDHELLIPSIS
Windows NT, 2000: Truncate text and add ellipses to text which does not fit into the static control.

Constant Definitions

Const SS_BITMAP = &HE
Const SS_BLACKFRAME = &H7
Const SS_BLACKRECT = &H4
Const SS_CENTER = &H1
Const SS_CENTERIMAGE = &H200
Const SS_ENDELLIPSIS = &H4000
Const SS_ENHMETAFILE = &HF
Const SSETCHEDFRAME = &H12
Const SSETCHEDHORZ = &H10
Const SSETCHEDVERT = &H11
Const SSGRAYFRAME = &H8
Const SSGRAYRECT = &H5
Const SSICON = &H3
Const SSLEFT = &H0
Const SSLEFTNOWORDWRAP = &HC
Const SSNOPREFIX = &H80
Const SSNOTIFY = &H100
Const SSOWNERDRAW = &HD
Const SSPATHELLIPSIS = &H8000
Const SSREALSIZEIMAGE = &H800
Const SSRIGHT = &H2
Const SSRIGHTJUST = &H400
Const SSIMPLE = &HB
Const SSUNKEN = &H1000
Const SSWHITEFRAME = &H9
Const SSWHITERECT = &H6
Const SSWORDELLIPSIS = &HC000

Back to the Window Class list.
Back to the Reference section.
Winsock Error Codes

The following list identifies the Winsock error codes. These values are used by the `WSAGetLastError` function to report and identify errors caused by other Winsock functions.

The following Winsock error codes are listed in ascending numerical order.

- **WSAEINTR (10004)**
  - Interrupted Function Call -- A blocking operation was cancelled.

- **WSAEACCESS (10013)**
  - Permission Denied -- An attempt to access a socket was forbidden by its access permissions.

- **WSAEFAULT (10014)**
  - Bad Address -- An invalid pointer address was specified in a function call.

- **WSAEINVAL (10022)**
  - Invalid Argument -- An invalid argument was passed to a function.

- **WSAEFILE (10024)**
  - Too Many Open Files -- There are too many open sockets.

- **WSAEWOULDBLOCK (10035)**
  - Resource Temporarily Unavailable -- The specified socket operation cannot be completed immediately, but the operation should be retried later.

- **WSAEINPROGRESS (10036)**
  - Operation Now in Progress -- A blocking operation is currently executing.

- **WSAEALREADY (10037)**
  - Operation Already in Progress -- An operation was attempted on a non-binding socket that already had an operation in progress.

- **WSANOTSOCK (10038)**
  - Socket Operation on Non-Socket -- An operation was attempted on something that is not a socket.

- **WSAENOTREACH (10039)**
  - Destination Address Required -- A required address was omitted from a socket operation.

- **WSAEVALNOTEQUAL (10040)**
  - Message Too Long -- A message was sent on a datagram socket that exceeds the internal message buffer or some other limit.

- **WSAEPROTOOPT (10041)**
  - Protocol Wrong Type for Socket -- A protocol was specified that is not supported by the target socket.

- **WSANOPROTOOPT (10042)**
Bad Protocol Option -- An unknown, invalid, or unsupported protocol option or feature was specified.

**WSAEPROTOONOSUPPORT (10043)**
Protocol Not Supported -- The specified protocol is not supported or is not implemented.

**WSAESOCKTNOSUPPORT (10044)**
Socket Type Not Supported -- The specified socket type is not supported in the address family.

**WSAEOPNOTSUPP (10045)**
Operation Not Supported -- The specified operation is not supported by the referenced object.

**WSAEPFNOSUPPORT (10046)**
Protocol Family Not Supported -- The specified protocol family is not supported or is not implemented.

**WSAFAFNOSUPPORT (10047)**
Address Family Not Supported by Protocol Family -- An address incompatible with the requested network protocol was used.

**WSAADDRINUSE (10048)**
Address Already in Use -- An attempt to use the same IP address and port with two different sockets simultaneously was made.

**WSAADDRNOTAVAIL (10049)**
Cannot Assign Requested Address -- The requested address is not valid (given the context of the function).

**WSAENETDOWN (10050)**
Network is Down -- A socket operation encountered a network that is down.

**WSAENETUNREACH (10051)**
Network is Unreachable -- A socket operation encountered an unreachable network.

**WSAENETRESET (10052)**
Network Dropped Connection on Reset -- A connection was broken due to "keep-alive" activity detecting a failure.

**WSAECONNABORTED (10053)**
Software Caused Connection Abort -- A connection was aborted by software on the host computer.

**WSAECNNECTION (10054)**
Connection Reset by Peer -- A connection was forcibly closed by the remote host.

**WSAENOBUFS (10055)**
No Buffer Space Available -- A socket operation could not be performed because the system ran out of buffer space or the queue was full.

**WSAEISCONN (10056)**
Socket is Already Connected -- A connect request was made on a socket that is already connected.

**WSAENOTCONN (10057)**
Socket is Not Connected -- An attempt to send or receive data failed because the socket is not connected.

**WSAENOTSHUTDOWN (10058)**
Cannot Send After Socket Shutdown -- An attempt to send or receive data failed because the socket has already been shut down.
WSAETIMEDOUT (10060)
  Connection Timed Out -- The remote host failed to respond within the timeout period.

WSAECONNREFUSED (10061)
  Connection Refused -- The target machine actively refused the attempt to connect to it.

WSAHOSTDOWN (10064)
  Host is Down -- The destination host is down.

WSAHOSTUNREACH (10065)
  No Route to Host -- The destination host is unreachable.

WSAEPROCLIM (10067)
  Too Many Processes -- The Winsock implementation has exceeded the number of applications that can use it simultaneously.

WSASYSTABLE (10091)
  Network Subsystem is Unavailable -- The underlying system to provide network services is unavailable.

WSAVERNOTSUPPORTED (10092)
  winsock.dll Version Out of Range -- The Winsock implementation does not support the requested Winsock version.

WSANOTINITIALIZED (10093)
  Successful WSAStartup Not Yet Performed -- The calling application has not successfully called WSAStartup to initiate a Winsock session.

WSAEDISCON (10094)
  Graceful Shutdown in Progress -- The remote party has initiated a graceful shutdown sequence.

WSATYPE_NOT_FOUND (10109)
  Class Type Not Found -- The specified class was not found.

WSAHOST_NOT_FOUND (11001)
  Host Not Found -- No network host matching the hostname or address was found.

WSATRY_AGAIN (11002)
  Non-Authoritative Host Not Found -- A temporary error while resolving a hostname occurred, and should be retried later.

WSANO_RECOVERY (11003)
  This is a Non-Recoverable Error -- Some sort of non-recoverable error occurred during a database lookup.

WSANO_DATA (11004)
  Valid Name, No Data Record of Requested Type -- The requested name is valid and was found, but does not have the associated data requested.

Constant Definitions

Const WSAEINTR = 10004
Const WSAEACCESS = 10013
Const WSAEFAULT = 10014
Const WSAEINVAL = 10022
Const WSAEMFILE = 10024
Const WSAEWOULDBLOCK = 10035
Const WSAEINPROGRESS = 10036
Const WSAEALREADY = 10037
Const WSAENOTSOCK = 10038
Const WSAEDESTADDRREQ = 10039
Const WSAEMSGSIZE = 10040
Const WSAEPROTOTYPE = 10041
Const WSAENOPROTOOPT = 10042
Const WSAEPROTONOSUPPORT = 10043
Const WSAESOCKTNOSUPPORT = 10044
Const WSAEOPNOTSUPP = 10045
Const WSAEPFNOSUPPORT = 10046
Const WSAEAFNOSUPPORT = 10047
Const WSAEADDRINUSE = 10048
Const WSAEADDRNOTAVAIL = 10049
Const WSAENETDOWN = 10050
Const WSAENETUNREACH = 10051
Const WSAENETRESET = 10052
Const WSAECONNABORTED = 10053
Const WSAECONNRESET = 10054
Const WSAENOBUFS = 10055
Const WSAEISCONN = 10056
Const WSAENOTCONN = 10057
Const WSAESHUTDOWN = 10058
Const WSAETIMEDOUT = 10060
Const WSAECONNREFUSED = 10061
Const WSAEHOSTDOWN = 10064
Const WSAEHOSTUNREACH = 10065
Const WSAEPROCLIM = 10067
Const WSASYSTIMEOUT = 10091
Const WSATYPE_NOT_FOUND = 10109
Const WSAHOST_NOT_FOUND = 11001
Const WSATRY_AGAIN = 11002
Const WSANO_RECOVERY = 11003
Const WSANO_DATA = 11004

Back to the Reference section.
The following is a list of various articles about the Windows API. Each article covers a single subject and continues over multiple pages. These articles are frequently written to provide a sort of overview for a set of API functions without going into the messy details of each individual function. (Of course, the specifics are only a click away in the reference section.) Choose a link below to go to that article's table of contents. Articles marked with NEW have been added to the list since the last update of the site.

- Faking 64-bit Integers
- Beginner's Guide to the Windows API

For more articles about the API, visit the API section of the VB-World.net web site.

Go back to the Windows API Guide home page.

Last Modified: April 16, 2000
This page is copyright © 2000 Paul Kuliniewicz. Copyright Information Revised October 29, 2000
Go back to the Windows API Guide home page.
E-mail: vbapi@vbapi.com Send Encrypted E-Mail
This page is at http://www.vbapi.com/articles/index.html
Faking 64-bit Integers

Introduction

Usually, the intrinsic Visual Basic data types are sufficient to read information from and send information to API functions. The two most important and frequently used data types are String and Long, which is a 32-bit integer data type. Since Windows is a 32-bit operating system, it almost exclusively uses 32-bit integers throughout its API.

However, 32 bits of integer data are not always enough. For some functions, a 64-bit integer data type is necessary to encompass the entire range of possible information. The ULARGE_INTEGER structure used by the API stores such a 64-bit integer by separating it into high-order and low-order halves of 32 bits each. This format is, sadly, not convenient for actually reading the data in the structure. To easily read and write to these structures, a 64-bit integer data type is necessary. And alas, Visual Basic has no true 64-bit integer data type.

The Currency Data Type

Fortunately, all is not lost! Visual Basic has the Currency data type. The Currency data type was initially created by the designers of Visual Basic to allow precise computations to be made on monetary values. The Single and Double floating-point data types are insufficient because they can only support a limited number of significant digits. Beyond that number of digits, some rounding occurs. The Currency data type combines the exactness of integer data types with a few fixed decimal places to allow subdivisions of a currency unit (such as a cent).

Internally, the Currency data type actually is a 64-bit integer. However, Visual Basic scales down by a factor of 10,000 to produce four digits after the decimal point. So, although the data type is a 64-bit integer, Visual Basic will display the value of any Currency-type variable as having a decimal point followed by four digits. Therefore, in order to display a 64-bit value copied into the variable correctly, you must first multiply the variable by 10,000. This will shift the decimal point four places to the right, resulting in the display of the actual value.

Example

Here is an example of how the Currency data type can be used to display a 64-bit integer retrieved from an API function. The GetDiskFreeSpaceEx reports all of its data as 64-bit integers. For disk space, 32-bit integers are too small; otherwise, the function would be limited to hard drives having a total storage space of less than 4.0 GB. Clearly, this is too small for modern disks!

After retrieving the information from the function, the following code copies it into a Currency-type variable. Then, after multiplying it by 10000 to move Visual Basic's decimal point, the free space of drive C: is properly stated.

Dim userbytes As ULARGE_INTEGER  ' bytes free to user
Dim totalbytes As UINTEGER  ' total bytes on disk
Dim freebytes As UINTEGER  ' free bytes on disk
Dim tempval As Currency  ' display buffer for 64-bit values
Dim retval As Long  ' return value of function

' Get information about the C: drive.
retval = GetDiskFreeSpaceEx("C:\", userbytes, totalbytes, freebytes)
' Copy totalbytes into the Currency data type.
CopyMemory tempval, totalbytes, 8
' Multiply by 10000 to move Visual Basic's decimal point to the end of the actual number.
tempval = tempval * 10000
' Display the total number of bytes on C:.
Debug.Print "Free Space on the C: drive:"; tempval; "bytes"

Manipulating 64-bit Integers

Although you may be storing a true 64-bit integer inside a Currency data type, Visual Basic believes there is a decimal point inside the value whether you want one there or not. Therefore, you must take care when performing some arithmetic operations on true 64-bit values. This involves moving the decimal point to preserve the actual integer, instead of the value Visual Basic believes you are storing inside of it.

Addition and Subtraction

No conversion factor is necessary to add or subtract 64-bit integers from one another. Simply add or subtract Currency types as you would any other data type values.

Multiplication

When multiplying two 64-bit integers in Currency form, you must multiply the result by 10,000 to move the decimal point to the proper place. However, if you wait to multiply by 10,000 until after you calculate the initial product, it is very possible that you will have lost part of the calculation due to internal rounding off. Therefore, to multiply two 64-bit integers in Currency form, use the following formula:

\[
\text{product} = 10000 \times \text{first} \times \text{second}
\]

Division

Like multiplication, division of two 64-bit integers in Currency form requires a correction factor. However, in this case, the quotient must be divided by 10,000. To minimize internal rounding off of significant digits, you should divide by 10,000 only after calculating the initial quotient. To divide two 64-bit integers in Currency form, use the following formula:

\[
\text{quotient} = \text{dividend} / \text{divisor} / 10000
\]

Limitations
Although using the Currency data type to "fake" 64-bit integer support in Visual Basic should work in most cases, there remains the problem of data range. Because the value must be multiplied by 10,000 to display properly, values at the extremes of the 64-bit integer range will cause an Overflow error if you try to display them. Thus, the Currency form of 64-bit integers in Visual Basic has a display range from -372,036,854,775,808 to 372,036,854,775,807. So while you cannot use the full range of 64-bit values, nevertheless these fake 64-bit integers greatly extend the range offered by the intrinsic 32-bit data type.

Of course, this limitation does not apply to 64-bit integers that do not need to be displayed for the user. The Currency data type itself can store the entire 64-bit range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Note, however, that at these extremes, the methods presented earlier for multiplying and dividing these values may fail. They depend on having the "extra room" available in the storage space that may not be present past the trillions. Again, however, despite these limitations, using the Currency data type as a pseudo-64-bit integer data type makes it significantly easier to use those API functions that, like GetDiskFreeSpaceEx, use 64-bit values in their parameters.
Abstract: In order to use the Windows API effectively in your programs, you must first learn a number of fundamentals about how the API operates. This article illustrates the usage of functions, structures, named constants, callback functions, and messages to empower hundreds of API functions. Other API-based ideas such as handles and device contexts also receive mention. Finally, important issues about the actual implementation of direct API calls in Visual Basic is mentioned.

Table of Contents

1. Introduction to the Windows API
2. Basics of Using API Functions
3. Concepts in the Windows API
4. Using API Structures
5. Using API Callback Functions
6. Advanced API Topics

Go back to the Articles section index.
Go back to the Windows API Guide home page.
Introduction to the Windows API

What is an API?

The acronym API stands for Application Programming Interface. It serves as a software interface to be used by other programs. Just as a number keypad is the interface for a calculator, an API is the software interface for things such as operating systems. The API of an operating system is the fundamental reason why different operating systems are incompatible -- why, for example, a Macintosh-based piece of software cannot run on a system running Windows 98 (without an emulator, at least). Because Mac OS and Windows have completely different APIs, programs designed to use one cannot use the other. The core function of an emulator is "converting" the API calls of one program into the API calls of the operating system running it; it acts as a sort of translator between the two APIs.

An API is truly the fundamental level of higher-level programming. In high-level programming, a program often does not execute tasks itself but instead tells some other program to do it. In the case of operating systems, programs frequently delegate various tasks to the underlying operating system. For example, if a program wishes to write data to a disk, it does not manually send commands to the hard disk itself to seek, read, write, etc. bit by bit. Instead, is merely tells the operating system to write a certain amount of data to a specified file, and the operating system handles the dirty work. The advantages of this arrangement are obvious. Designers of software do not have to worry about the basic, fundamental chores involved in every program (such as disk access, memory allocation, drawing on the monitor, etc.) by hand, saving both time and final program size. Also, suppose that a more efficient way to render an image on a monitor is created. If each individual program directly handled all its graphical output, each program would need to be updated to take advantage of this development. Instead, if they all use the API, and the operating system is upgraded to include this new method, every program now enjoys the benefits of the more efficient system without a single modification.

What Does the Windows API Do?

Basically, the Windows API handles everything that makes Windows what it is. Naturally, it encompasses aforementioned things such as drawing on the monitor, disk access, printer interface, Internet usage, etc. However, it also provides most of the features common to all Windows-based software. For example, the common dialog boxes (Open, Save As, Choose Font, etc.), the Windows shell, list boxes, operating system settings, and even windows themselves are provided by the Windows
API. Windows-based programs make extensive use of the Windows API in almost every task. Even if you don't use the API explicitly in your programming, the programming language software will almost always put calls to the Windows API in the generated program to handle various tasks. The Windows API in all encompasses a surprising vast number of topics.

**Where is the Windows API?**

Almost every function in the Windows API is located in one of the DLL (Dynamic Link Library) files found in the C:\Windows\System directory (or wherever the Windows System directory is). These DLL files "export" functions inside of them -- they allow external programs to use their functions. This design allows any Windows-based program to access any API function relatively easily. The bulk of API functions are found in user32.dll (user interface functions), kernel32.dll (operating system kernel functions), and gdi32.dll (graphics device interface functions), and shell32.dll (Windows shell functions).

**What are the Components of the Windows API?**

Although so far on this page only the API functions have been refered to, the functions themselves are only one part of the entire API. The following list identifies all the objects which make up the Windows API.

- **Functions** - As already mentioned, the functions make up the core of the Windows API. They are the actual code which accomplish the various tasks handled by the API. They are stored in the DLL files and can be accessed easily from any Windows-based program.

- **Structures** - Structures, combinations of multiple individual variables, are frequently used in the Windows API to store groups of related information. Many API functions require that a structure be passed to them in order to more conveniently transfer a large amount of information without an overly large number of parameters. Although these structures are used by the API functions, your program must properly define them itself.

- **Named Constants** - The Windows API frequently uses otherwise esoteric numeric codes for information. Naming these constants provides a more convenient way to refer to these values in code. Sometimes these constants are used as flags: the data value can be a binary OR of multiple individual flags, allowing one value to store multiple pieces of information. As with structures, the named constants must be explicitly defined in your program.

- **Callback Functions** - Conceptually, a callback function is the reverse of an API function. A callback function is defined completely in your program. This function is then called by an API function routine when it executes its task. Callback functions provide a way to have your program play a critical role in a task. Callback functions are used frequently in enumerations, where all the members of some group are identified one by one; in this case, a callback function is used to allow your program to somehow process each item found belonging to the group.

- **Messages** - In a way, messages are a special type of named constant. While they are also numeric values given names, they behave differently. Messages are sent to objects (especially windows) to
tell them to do something. Sending a message does not in itself do anything; rather, it instructs some other object to execute one or more events.

Why Bother?

Granted, using the Windows API to do stuff does have drawbacks. First, API functions are significantly more error-prone than using intrinsic programming language functions. Because the entire API library was created in C++, the information passed to and from the functions must be compliant with C++ syntax, regardless of the programming language you are using. This can cause hard-to-find problems regarding pointers, terminating nulls, and other things you'd rather not have to deal with. Furthermore, API functions are prone to fail spectacularly. Instead of making a nice little error dialog appear, they can easily crash the entire program, forcing it to shut down with a GPF (General Production Fault). And since Visual Basic comes with almost no API documentation whatsoever, you need to find other resources (such as this site!) to learn what functions even exist, let alone how to use them. So why bother at all?

Simple: because API functions are also significantly more powerful than the intrinsic features of pretty much any programming language, especially Visual Basic. For example, in VB you can set the title bar text of any window you created using its .Caption property. But how do you change the text of a window not created by your program? The SetWindowText API function comes in handy, not to mention others. Another example: if you want to use the Open File common dialog box without the API, you need to distribute an OCX library of about 90KB along with your program. Using the GetOpenFileName API function, you can avoid the overhead be accessing what the OCX really serves as a front-end for. And if you want to, say, read and write the Windows registry or use the joysticks connected to the computer, Visual Basic doesn't have the necessary tools. However, even those become possibly using the Windows API!

So, in summary, if Visual Basic or whatever language you use has the ability to perform a certain task, you're better off not using the Windows API for that. You'll have easier requirements and the security of having a more reliable mechanism. But if you want to add more power to your program which VB can't provide, the Windows API is your only way to go. This guide endeavors to provide the Visual Basic user with the information and examples necessary to allow him or her to successfully implement hundreds of API functions (as necessary) in his or her programs.

Go back to the Articles section index.
Go back to the Windows API Guide home page.
Basics of Using API Functions

Introduction

On the surface, using a function from the Windows API in a Visual Basic program seems fairly simple and straightforward. (Don't worry -- we'll get to the complicated stuff soon enough!) Before you begin using the function, you must first declare it somewhere in your program. Then, you can use the function just as you would any other function in Visual Basic. This page explains how to declare a function and demonstrates some simple examples of how to use them in code.

Declaring the Function

Before an API function can be using in Visual Basic, it must first be declared. By declaring the function, you tell Visual Basic where it can find the function. The declaration identifies the name of the function, the DLL file it appears in, the parameters it requires, and the data type (if any) it returns. Armed with this information, Visual Basic knows where to find the API function whenever it is invoked. (This procedure isn't limited to Windows API functions. If you import any functions from a DLL file, they must be declared in the same manner. Using other DLL function libraries is not covered by this site.)

The below statement illustrates the syntax of the declaration. The Declare statement in Visual Basic is used to declare a function. The Declare statement can only appear in the (declarations) section of a form or module. If it appears in a form, the declaration must be Private, making the function only available inside that form. If it appears in a module, the declaration can be either Public or Private. The Public keyword makes the function available throughout the entire program, whereas the Private keyword restricts its use to within the module itself.

```vbnet
{{Public | Private}} Declare Function function_name Lib "DLL_filename" [Alias "function_alias"] (argument_list) As data_type
```

Since this syntax might seem intimidating at first, the following list identifies the various components of the Declare statement.

- **function_name**
  
  The name of the API function. This is the name by which Visual Basic will refer to the function
everywhere else in your code. In theory, you can make this any name you want (as long as you use the proper Alias clause), but it's safest to make this the "official" name of the function in the Windows API.

**DLL_filename**

The name of the DLL file which stores the function. This does not include the path, because the Windows System directory (where all API DLL files appear) is assumed. Actually, the filename does not even have to specify the .dll extension, so user32 and user32.dll are synonymous. However, watch out for some API functions which do not occur in DLL files; for example, some printer-using API functions appear in winspool.drv. In cases like that, the extension must be specified.

**function_alias**

(Optional) The name of the function as it appears in the DLL file. This is important because almost every function which has a string as a parameter has two versions: the ANSI version (used by most speaking languages) and the Unicode wide-character version (used by wide-character set languages such as Chinese). Since this web site is designed primarily for English speakers, the ANSI version of the functions are always used. When an ANSI/Unicode pair appear in a DLL, they are given slightly different names: the ANSI version ends with the letter A, and the Unicode version ends with the letter W. For example, the two versions of `CompareString` are called `CompareStringA` and `CompareStringW`.

**argument_list**

A list of the zero or more arguments which the function needs. This is very similar to any other argument list in Visual Basic, but its added features in the Declare statement will be discussed in the next section.

**data_type**

The name of the data type which the function returns. This will almost always be a Long, although a few function returns a String.

In case you were wondering, some functions do not return any value. For those functions, the Declare statement takes a slightly different format, presented below.

```
[{Public | Private}] Declare Sub function_name Lib "DLL_filename"
[Alias "function_alias"] (argument_list)
```

### The Argument List

The argument list specifies how many and what kinds of variables are passed to the function. Its format is very similar to the argument list of a program-defined Function or Sub. The following line illustrates the syntax of the argument list. Keep in mind that some functions take an empty argument list -- they are not passed any data whatsoever.

```
[{ByVal | ByRef}] argument_name As data_type, ...
```
Except for the option for ByVal and ByRef, which will be introduced shortly, the format of the argument list is similar to what you are probably used to already. But for review, here's what the different symbols in the argument list syntax represent.

**argument_name**

The name given to the argument. When declaring API calls, this really only serves to give a clue as to what the parameter represents. Although you should use the argument name suggested by this guide (or by other API documentation), you are actually free to choose whatever name you wish.

**data_type**

The data type of the individual argument. This could also be the "Any" keyword, which allows information in any data type to be passed.

### Data Types

As you already know, a data type specifies the size and format of a variable or argument to a function. API functions support only a limited number of data types compared to those available in Visual Basic. The following list identifies which data types can be used in API function declarations.

- **Byte**
  
  An 8-bit integer.

- **Integer**
  
  A 16-bit integer.

- **Long**
  
  A 32-bit integer.

- **String**
  
  A variable-length string.

In addition to the aforementioned data types, any of the API structures can also be used. Structures will be dealt with in a later page in this introduction. Out of the four data types mentioned above, Long and String are the most common. Since Windows is a 32-bit operating system (starting with Windows 95 and Windows NT 3.1), almost all numbers used in its API are Longs, 32-bit integers. Strings are also used frequently because they are pretty much the only way to store strings.

### ByVal and ByRef: An Introduction

The ByVal and ByRef keywords specify the method used to pass a parameter to the API function. These two methods are extremely different and nonsubstitutable — you can never use ByRef in place of ByVal and vice versa. When declaring API functions, the ByRef method is assumed to be used unless the ByVal keyword is used explicitly. Therefore, in API function declares, you will usually never see ByRef explicitly used; it will only be apparent by the lack of the ByVal keyword.
The actual meanings of ByVal and ByRef are beyond the scope of this individual page, although knowledge of their actual interpretations is crucial for using the more sophisticated API functions. At this point, however, the gist of these two keywords can be summarized as follows.

**ByVal** literally means "By Value." The value of whatever variable or expression is passed as the parameter. This method prevents the function from altering the contents of a variable passed ByVal. For example, if you have a variable MyVar that you pass ByVal to an API function, the API function cannot possibly edit the contents of MyVar.

**ByRef** literally means "By Reference." Instead of passing the contents of the variable, this method passes a sort of reference to the variable itself. This allows the function to edit the contents of the variable passed as that parameter. Note that expressions or constants cannot be passed ByRef -- only variables. For example, a variable MyVar passed ByRef to an API function may very well be edited by that function. In fact, ByRef parameters are usually used to allow the function to "return" information outside its single return value.

There are a few additional points to remember about ByVal and ByRef:

1. Strings are always passed ByVal. However, the API function is able to edit the contents of the string variable, as if it had been passed ByRef. The reasons for this are beyond the scope of this page.
2. User-defined structures, including structures used by the API, are always passed ByRef.
3. Arrays are always passed ByRef when the entire array is meant to be passed to a function. When passing an entire array to a function, VB syntax mandates that you "pass" the first element of the array to the function. However, in reality you are passing the entire array if that parameter is defined to by ByRef. If the parameter is instead ByVal, passing the first element of the array does just that.
4. Numeric variables can be passed either ByVal or ByRef, depending on the requirements of the function.

**Examples**

This page has presented a lot of information about the Declare statement in Visual Basic. At first, using Declare may seem very difficult, especially since I am saving some of the uglier details for a later page of this API introduction series. Fortunately, you probably will never write a Declare yourself. You can get the Declares for API functions from the API Text Viewer that comes with Visual Basic, or you can get the Declares off the pages of this site. With these resources, declaring API functions becomes just a copy-and-paste operation. Nevertheless, a number of examples of the Declare statement will help you understand the contents of this page.
The following list presents a number of examples of the Declare statement as well as a brief explanation of what they mean. Don't worry about what these functions actually do at this point, although you can read about any of them elsewhere in this site. What's important here is understanding the mechanics of the Declare statement.

- The following line declares a function which does not take any parameters and returns a 32-bit integer:
  
  ```vba
  Declare Function GetDesktopWindow Lib "user32.dll" () As Long
  ```

- The following line declares a function which takes a single 32-bit parameter by value and does not have a return value:

  ```vba
  Declare Function Sleep Lib "kernel32.dll" (ByVal dwMilliseconds As Long)
  ```

- The following line declares a function which takes two 32-bit parameters by value and returns another 32-bit integer:

  ```vba
  Declare Function ReleaseDC Lib "user32.dll" (ByVal hWnd As Long, ByVal hDC As Long) As Long
  ```

- The following line declares a function which takes one string as a parameter and returns another string. Notice how this function, like almost all string-using API functions, requires an alias:

  ```vba
  Declare Function CharUpper Lib "user32.dll" Alias "CharUpperA" (ByVal lpsz As String) As String
  ```

- The following line declares a function which takes one RECT structure as a parameter and returns a 32-bit integer:

  ```vba
  Declare Function CreateRectRgnIndirect Lib "gdi32.dll" (lpRect As RECT) As Long
  ```

- The following line declares a function which takes three parameters. The first two parameters are 32-bit integers passed by value. The third parameter is a 32-bit integer passed by reference -- that variable's value is set by the function. The function also returns a 32-bit integer:

  ```vba
  Declare Function SHGetSpecialFolderLocation Lib "shell32.dll" (ByVal hwndOwner As Long, ByVal nFolder As Long, ppidl As Long) As Long
  ```

- Finally, this declaration encompasses a number of different ideas simultaneously:

  ```vba
  Declare Function RegCreateKeyEx Lib "advapi32.dll" Alias "RegCreateKeyExA" (ByVal hKey As Long, ByVal lpSubKey As String, ByVal Reserved As Long, ByVal lpClass As String, ByVal dwOptions As Long, ByVal samDesired As Long, ByVal lpSecurityAttributes As SECURITY_ATTRIBUTES, phkResult As Long, lpdwDisposition As Long) As Long
  ```

That completes this section of the Introduction to the Windows API. If that last example declaration confuses you, you should review the contents of this page to cover any ideas which are still fuzzy. Remember, the rest of this series of pages builds on the fundamentals outlined above.
Concepts in the Windows API

Introduction

Although this series has already discussed the mechanics of declaring and using API functions, it has not yet dealt with some basic concepts used throughout the Windows API. Few of these ideas appear explicitly in Visual Basic, although their knowledge and mastery is crucial for using the API effectively. This page identifies these important ideas, explains them, and briefly demonstrates where they can appear.

Handles

Handles are internal constructs of the Windows API. The API uses handles to keep track of a large number of objects throughout the system. For example, open files, windows, open registry keys, brushes, icons, menus, and many other objects are referred to using handles. In fact, their handles are really the only way a program is able to manipulate these objects, since Windows itself handles all the little details.

In reality, a handle refers to an internal structure in Windows detailing various information and properties of the object. However, there is no way to access this structure directly; that's why it is internal! The handle itself is just a 32-bit integer used by a program calling API functions. In Visual Basic, the Long data type is used to store handles. As far as Visual Basic is concerned, there is no special difference between a handle and any other 32-bit integer value. The significance of a handle only becomes apparent when it is passed as a parameter to an API function.

As already mentioned, most objects under Windows have handles attached to them. This includes form windows, command buttons, picture boxes, and many other objects which Visual Basic can create and manage for you. Fortunately, these objects, when created using Visual Basic, have a property called hWnd. This property equals the handle to that window. So, for example, if you have a command button called Command1, you can use the property Command1 hWnd to get the handle to the command button (remember that buttons, forms, etc. are all just types of windows). This handle can be given to any API function which can use it properly.

The following example demonstrates how handles can be used. The example uses the API to draw the first icon stored in the file C:\Windows\sol.exe. This example uses three handles. The first handle, App.hInstance, is merely a handle to this particular instance of the example program. The second handle, called hIcon, is a handle to the icon used to depict the icon. The third handle, Form1.hDC, is a handle to Form1's device context. Device contexts are explained in the next section.

Dim windir As String  ' receives path of Windows directory
Dim slength As Long  ' receives length of Windows directory path
Dim exename As String  ' full filename of sol.exe
Dim hIcon As Long  ' handle to the display icon
Dim retval As Long  ' generic API function return value
' First, since the Windows directory might not be C:\Windows,
' obtain the actual path of it.
windir = Space(256)  ' make enough room in the buffer
' Place the directory name in the buffer.
slength = GetWindowsDirectory(windir, 256)
' Remove the blank space from the buffer.
windir = Left(windir, slength)
' Combine the Windows directory path and the filename.
exename = windir
If Right(exename, 1) <> "\" Then exename = exename & "\"
exename = exename & "sol.exe"

' Now, extract the first icon stored in the Solitaire program.
hIcon = ExtractIcon(App.hInstance, exename, 0)
' Draw this icon in the upper-left corner of Form1.
retval = DrawIcon(Form1.hDC, 0, 0, hIcon)
' Destroy the loaded icon to free resources.
retval = DestroyIcon(hIcon)

Device Contexts

Device contexts have a similar initial appearance to handles. Like a handle, a device context is actually an internal data structure inside Windows which cannot be accessed by your program. As the name suggests, a device context is an intermediary between your program and a physical device, such as a display monitor, the keyboard, or the printer. Device contexts are important because they allow your program to treat different models and brands of one type of device (say, printers) in identical ways. By using device contexts, your program doesn't care if the user has an HP DeskJet or a Canon BubbleJet because the device contexts for both are almost identical.

As already mentioned, a program cannot access the device context of a device directly. Instead, it can only use a handle to the device context. It is important to remember that the device context and a handle to a device context are not the same thing! A device context is not accessible directly by a program. A handle to a device context is, like any other handle, a 32-bit integer.

Interestingly, windows themselves are considered to be devices in the sense that they can have a device context. This allows graphics API functions to draw on and otherwise manipulate the appearance of a window. Fortunately again, Visual Basic automatically assigns the .hDC property of such windows to the handle to that window's device context. This allows calls to API functions to reference VB-created objects relatively easily.

The following example demonstrates how a device context can be used. Here, a rounded rectangle is drawn on the form window Form1. The rounded rectangle is drawn with a thin white pen and is filled with a green diagonally cross-hatched brush. Notice that whenever an API function needs to do something to Form1, it requires the handle to its device context (Form1.hDC). You can also see those all-important handles used to identify the brush and pen, not to mention the device context!

Dim hPen As Long  ' handle to the pen
Dim hBrush As Long  ' handle to the brush
Dim hOldPen As Long  ' handle to Form1's previous pen
Dim hOldBrush As Long  ' handle to Form1's previous brush
Dim retval As Long  ' generic API return value

' (Recall that all constant definitions appear on the pages which
' explain the function they are used by. So here, the constant
' WHITE_PEN is defined on GetStockObject's page.)

' Get a handle to the desired pen.
hPen = GetStockObject(WHITE_PEN)

' Get a handle to the desired brush.
hBrush = CreateHatchBrush(HS_DIAGCROSS, RGB(0, 255, 0))

' Select the new pen and brush for use by Form1.
hOldPen = SelectObject(Form1.hDC, hPen)
hOldBrush = SelectObject(Form1.hDC, hBrush)

' Draw the rounded rectange on Form1.
retval = RoundRect(Form1.hDC, 100, 100, 300, 200, 5, 5)

' Select the previous pen and brush for Form1 and delete
' the ones created by this example.
retval = SelectObject(Form1.hDC, hOldPen)
retval = SelectObject(Form1.hDC, hOldBrush)
retval = DeleteObject(hPen)
retval = DeleteObject(hBrush)

## Pointers

A pointer is simply a 32-bit integer variable which holds a memory address. This memory address is usually the location of some other object, such as a structure, another variable, or a structure. Despite this deceptively simple definition, pointers are a core component of computer programming, especially in the C++ language. Interestingly, Visual Basic has no intrinsic support for pointers whatsoever (well, that's not quite true, but the exception will come in a later page of this series), although you can use the Long data type to store a pointer. Presumably, VB has no explicit pointers to make programming easier. (A later page in this series will illustrate some places where Visual Basic hides the pointers it uses.)

Because the DLLs which constitute the Windows API were built using some flavor of C++, many of them desire pointers to one object or another as one or more of their parameters. At this time, it is sufficient to say that 99.5% of the time Visual Basic successfully relieves the burden of creating all the otherwise necessary pointers to structures and strings and other variables. However, the other .5% of the time, your code will need to generate its own pointers, usually by using other API functions.

The following example demonstrates one instance where your program ought to manipulate a pointer explicitly. Here, a PIDL -- a pointer to an ITEMIDLIST structure -- is used to help display the icon used for the Favorites folder in the Windows system shell. Because the API is kind enough to create and delete ITEMIDLIST structures itself, there is no need to access the structure directly; in fact, doing so will almost always generate some error. Therefore, the example blissfully uses a pointer to such a structure. (Once again, handles and device contexts pop up in this example. See how fundamental they are?)

If you have problems understanding this example, you might want to skip ahead to the page of this series about structures,
since this example uses one structure. I couldn't think of any good examples of pointers that didn't necessitate using one
structure or another.

Dim pidl As Long  ' PIDL to the Favorites folder
Dim fileinfo As SHFILEINFO  ' stores file information
Dim retval As Long  ' generic return value

' Get a PIDL (pointer to an ITEMIDLIST) which refers to the
' Favorites folder in the Windows shell.  (pidl is the PIDL)
retval = SHGetSpecialFolderLocation(Form1.hWnd, CSIDL_FAVORITES, pidl)

' Now get a little info about the Favorites folder.  Specifically, the following
' function call merely requests the folder's icon.
retval = SHGetFileInfo(pidl, 0, fileinfo, Len(fileinfo), SHGFI_ICON Or SHGFI_PIDL)

' The handle fileinfo.hIcon now refers to the Favorites folder's
' icon in the shell.  Display it on Form1.
retval = DrawIcon(Form1.hDC, 0, 0, fileinfo.hIcon)

' Destroy the icon to free resources.
retval = DestroyIcon(fileinfo.hIcon)

Footnote: Combining Flags

In the call to SHGetFileInfo in the above example, you see your first exposure to flags. A flag is simply a type of named
constant. The special thing about flags is that they can be combined with other related flags. This allows multiple on/off or
yes/no options to be passed to a function using a single parameter. Above, you see that the function is told to retrieve a
handle to the icon (SHGFI_ICON) and that the first parameter is a PIDL to the folder (SHGFI_PIDL).

When combining multiple flags, the bitwise Or operator should always be used. This assures an error-free combination of
the flags (as long as the attempted combination is valid). Why not use the addition operator (+) to combine the flags.
Although the effect may be the same, using regular addition can have some unexpected consequences. To demonstrate the
problem which can appear when using addition, I will create my own hypothetical function.

Consider a message-box-like function which accepts a single parameter. This parameter is a combination of the following
flags specifying the buttons you want to appear in the box:

Const PK_OK = 1 ' OK button only
Const PK_CANCEL = 2 ' Cancel button only
Const PK_OKCANCEL = 3 ' OK and Cancel buttons
Const PK_HELP = 4 ' Help button only

Remember that this is a hypothetical example -- these flags do not exist in the API. You will notice that the
PK_OKCANCEL flag is actually the bitwise Or combination of the PK_OK and PK_CANCEL flags (1 Or 2 = 3) and not
a separate flag in itself. This is done to simplify things for the programmer, since the OK and Cancel buttons frequently
appear together.

Now suppose that a novice API programmer decides he wants to use the OK and Cancel buttons. At first, he sends merely
the PK_OK flag to the function:
retval = ExampleFunction(PK_OK)

Accordingly, only the OK button appears in the dialog box. Realizing his mistake, he decides to fix his code by adding the PK_OKCANCEL flag as one of the flags. So he enters the following:

retval = ExampleFunction(PK_OK + PK_OKCANCEL)

And he expects both the OK and Cancel buttons to appear. But instead the Help button appears by itself! Why? Notice that PK_OK + PK_OKCANCEL = PK_HELP (i.e., 1 + 3 = 4). His expression actually evaluated to the value for the PK_HELP flag by itself! Had he used the bitwise Or operator has follows, he would have entered:

retval = ExampleFunction(PK_OK Or PK_OKCANCEL)

That line executes as expected, displaying the OK and Cancel buttons. Why? Notice that PK_OK Or PK_OKCANCEL = PK_OKCANCEL (i.e., 1 Or 3 = 3). Using Or instead of addition guarantees that accidentally specifying the same flag twice will not cause an error. Of course, it is sloppy coding, so you should avoid doing so anyway. Of course, in the "real" world this error might not be so obvious. (For example, see the SetWindowPos function; the SWP_DRAWFRAME and SWP_FRAMECHANGED flags are actually identical, and adding the two together gives you the SWP_SHOWWINDOW flag instead.)
Using API Structures

Introduction

This page begins the discussion of another key component of the Windows API: the structure. Structures allow a function to receive or return a large amount of information without cluttering the argument list. Structures almost always group related information, allowing related data to remain in a single package. The usage of structures is important in the more complex (and therefore more useful) API functions.

Structure Fundamentals

A structure is an object used in programming to group multiple related variables. A structure consists solely of one or more individual variables having any data type (including that of another structure). To use a structure, create a variable having that structure as its data type. This structure variable now has one of each member of the structure inside of it. Usually these data members store information about a related thing.

Structures in Visual Basic

To define a structure in Visual Basic, the Type block is used. Structure definitions can only appear in the (declarations) section of a form or module. The Type block has the following syntax:

```
[[Public  |  Private}] Type type_name
    member1 As data_type1
    member2 As data_type2
    ...
End Type
```

The Public/Private indicator works the same way as it does for functions. Structures are public unless otherwise specified. The following list identifies the various components of the Type block:

- **type_name**
  The name to give the structure. This name will then be used as the data type of whatever variables you want to use the structure.

- **member1, member2, ...**
  The name of an individual member of the structure. These are effectively variables stored in the
structure.

data_type1, data_type2, ...

The data type of a particular item in the structure. Valid types include the Byte, Integer, Long, and String data types. This could also be another structure (although of course not the one being defined). This could also be a fixed-length string, having the data type of String * numchars, where numchars is the size in characters of the fixed-length string.

For example, consider the following hypothetical structure:

```
Type EXAMPLESTRUCT
  longvar As Long
  another As Long
  s As String
  fixed As String * 24
  astruct As RECT
End Type
```

In this example, the structure has five data members. longvar and another are both 32-bit integers. s is a regular string, but fixed is a string which has a fixed length of 24 characters. astruct is a variable of the RECT structure type.

To access a data member of the structure (after you have defined a variable to use the structure), simply use the . (period) operator between the variable name and the member name. Apart from that, the data members inside the variable can be treated exactly as a regular variable. For example, all of the following code is perfectly valid:

```
Dim ex As EXAMPLESTRUCT  ' ex is the structure
ex.longvar = 54
ex.another = ex.longvar * 65 - 9
ex.s = "This is a sample string."
ex.fixed = "Fixed-length string."  ' this will be right-padded with spaces
Debug.Print ex.fixed
ex.astruct.left = 54
```

### API Structures

Many functions in the Windows API use structures as one or more of their arguments. This allows large amounts of related information to be passed to and from the function relatively easily. Although some structures are used by dozens of different functions, you cannot link to a structure in any of the API DLL files. Instead, you must define any API structures you use yourself in your program. Naturally, you must use the exact same definition of the structure in your code, or else fatal errors are likely to appear. For example, if you wish to use the RECT structure in your program, the following definition must also appear in your program:
Here, the **RECT** structure provides a convenient way to keep the necessary coordinates of a rectangle grouped together. Each member of this particular structure holds one component of one of the two coordinate pairs used to identify the rectangle. Without this structure, four individual variables would be necessary to hold the same information. This structure is commonly used in graphics API routines.

Still not convinced about the usefulness of structures? This example taken from the API should be able to persuade you. There are two functions that create a font object: **CreateFont** and **CreateFontIndirect**. Both take identical information and perform the same operations to create the desired font. However, one accepts a **LOGFONT** structure to receive all the necessary information, whereas the other needs each member of the structure to be passed individually. Below are the declarations for each function.

```vba
Declare Function CreateFontIndirect Lib "gdi32.dll" Alias "CreateFontIndirectA" (lplf As LOGFONT) As Long

Declare Function CreateFont Lib "gdi32.dll" Alias "CreateFontA" (ByVal nHeight As Long, ByVal nWidth As Long, ByVal nEscapement As Long, ByVal nOrientation As Long, ByVal fnWeight As Long, ByVal fdwItalic As Long, ByVal fdwUnderline As Long, ByVal fdwStrikeOut As Long, ByVal fdwCharSet As Long, ByVal fdwOutputPrecision As Long, ByVal fdwClipPrecision As Long, ByVal fdwQuality As Long, ByVal fdwPitchAndFamily As Long, ByVal lpszFace As String) As Long
```

Clearly, the use of structures greatly simplifies the use of some functions, especially when large amounts of information need to be passed to or from the function. Notice that when a structure is passed to a function, it is always passed ByRef. Structures are generally easy to use in the API and should not cause any great problem using them.
Using API Callback Functions

Introduction

Callback functions are a powerful tool, giving great flexibility to some API functions. A callback function allows your program to build its own routines to handle events generated by the API functions themselves. Your program must contain any callback functions it wishes to use, because Windows does not define any "default" callback functions.

Usage of Callback Functions

Callback functions are used when it is necessary that the program calling the API function handle some of the work. The most common examples of callback functions occur in enumeration. During an enumeration, the invoked API function locates all objects which fit the desired category. For example, when using `EnumWindows` to enumerate all windows currently open, the API function obtains a handle to each window it finds. However, the API function does not know what to do with all the handles it finds. This is where the callback function comes in.

Callback functions typically process some sort of data during the middle of a call to an API function. Continuing the example, `EnumWindows` itself will call the program-defined callback function you specify and give it the handles, one at a time, that it finds. It is up to the callback function to decide to do whatever it wants with those handles. Whenever the callback function is finished, it is given its next handle until either all windows have been enumerated or the callback function instructs `EnumWindows` to stop.

Of course, enumerations are only one case where callback functions might be used. They can also be used to set up a message handler routine for an object such as a window. This allows a program to define its own custom message handling routines for some messages, passing the ones it ignores to the default handler used by Windows. Even more uses for callback function are possible. Generally, an API function will require an associated callback function whenever it must leave the implementation of some necessary task to the program calling the API function.

Callback Functions in Visual Basic

You may recall that in an earlier page of this series, I mentioned that Visual Basic has no explicit usage of pointers. I lied. Visual Basic does in fact have a single operator -- the AddressOf operator -- which allows the programmer to access one type of pointer. The AddressOf operator is absolutely necessary in order to use a callback function. Unfortunately, since AddressOf was first introduced in Visual Basic 5.0, any earlier versions of Visual Basic have no way whatsoever of using callback functions. No other workaround exists. In fact, the AddressOf operator was developed specifically to allow Visual Basic programs to use callback functions!

The AddressOf operator has very limited uses. Most importantly, it can only return the address of a function defined by your program. This function must be Public and be defined in a module (not a form). Furthermore, the AddressOf operator itself can only be used inside of the argument list of a call to a function; it cannot be used any other time. In use, the AddressOf operator has the following syntax:
AddressOf function_name

function_name is the name of the function to get the address of. AddressOf returns a pointer to the specified function. Any callback-using API function needs a pointer to the callback function in order to invoke it. Keep in mind that the AddressOf operator itself does not call the function given to it, and it does not take the argument list of that function either. It merely returns a pointer to where that function appears in memory.

Example

Below is a small example demonstrating how a callback function might be used. This example enumerates all of the windows currently open. For any windows which have a nonempty caption, the callback function prints that caption in the Debug window. You'll notice that the "meat" of this example is in the callback function itself. The other part of the code only serves to call the proper API function. Also note how various sections of the example must be copied into different places of a Visual Basic program.

' Display the title bar text of all top-level windows. This
' task is given to the callback function, which will receive each handle
' individually.
' Note that if the window has no title bar text, it will not be displayed (for
' clarity's sake).

' *** Place this code in a module. This is the callback function. ***
' This function displays the title bar text of the window identified by hwnd.
Public Function EnumWindowsProc (ByVal hwnd As Long, ByVal lParam As Long) As Long
Dim slength As Long, buffer As String  ' title bar text length and buffer
Dim retval As Long  ' return value
Static winnum As Integer  ' counter keeps track of how many windows have been
enumerated

winnum = winnum + 1  ' one more window enumerated....
slength = GetWindowTextLength(hwnd) + 1  ' get length of title bar text
If slength > 1  ' if return value refers to non-empty string
    buffer = Space(slength)  ' make room in the buffer
    retval = GetWindowText (hwnd, buffer, slength)  ' get title bar text
    Debug.Print "Window "; winnum; " : ";  ' display number of enumerated window
    Debug.Print Left(buffer, slength - 1)  ' display title bar text of enumerated
window
End If

EnumWindowsProc = 1  ' return value of 1 means continue enumeration
End Function

' *** Place this code wherever you want to enumerate the windows. ***
Dim retval As Long  ' return value

' Use the above callback function to list all of the enumerated windows. Note that
lParam is
' set to 0 because we don't need to pass any additional information to the function.
retval = EnumWindows (AddressOf EnumWindowsProc, 0)
Footnote: Working Around AddressOf's Limitations

The AddressOf operator is only valid when it appears in the argument list of a function when it is called. However, sometimes it is necessary to set a data member of a structure to the address of a function. Some API functions include a pointer to a callback function as part of the structure passed to them. However, something like the following will not work properly:

```
Dim ofn As OPENFILENAME
ofn.lpfnHook = AddressOf AnyFunctionName
```

Although you need to set ofn.lpfnHook to a pointer to the function AnyFunctionName, the AddressOf operator will cause an error. So how do you get the information into the structure's data member? Remember that AddressOf is valid only inside of a function call -- a function call in general, not necessarily a call to an API function! The trick is to create some sort of "dummy" function such as the one below.

```
Public Function DummyFunc (ByVal pointer As Long) As Long
    DummyFunc = pointer
End Function
```

This function simply returns whatever value was passed to it originally. So what is the purpose of this function? Take a look at the code below, which is a modified form of the original attempt to use AddressOf with a structure:

```
Dim ofn As OPENFILENAME
ofn.lpfnHook = DummyFunc(AddressOf AnyFunctionName)
```

This code executes perfectly! AddressOf here is valid because it appears in the argument list to DummyFunc. Since DummyFunc just returns whatever was passed to it, the end result is that ofn.lpfnHook is effectively set to AddressOf AnyFunctionName. The dummy function acts as a tool to work around AddressOf's limitations. You'll need to use DummyFunc or a similarly defined function whenever you want to use AddressOf outside of a function call.
Advanced API Topics

Introduction

As I mentioned earlier, Visual Basic hides some of the more difficult (and therefore powerful) concepts in programming from you. Its rationale is noble -- to make Visual Basic a simpler language to use -- and it often pays off. However, this simplicity becomes a liability when using the Windows API. The most obvious case here is pointers. Because Visual Basic almost always hides the pointers it uses from the programmer, using them in API functions can at times be difficult.

This final page reveals some of what Visual Basic hides from you during API-based programming. You won't find very much of this in Visual Basic's documentation; I learned most of this from experience and frustration. Nevertheless, a firm understanding of the following material is crucial for mastering the most complex of API functions.

The Truth About ByVal and ByRef

Earlier in this series, the superficial meanings of ByVal and ByRef were defined. However, in reality something somewhat more sinister is happening. You already know that Windows is a 32-bit operating system. Well, it just so happens that every parameter passed to any API function is a 32-bit integer.

"Wait just a minute," I'm sure you're saying right now. "You can too pass other things as parameters to API functions. Why, some functions take strings as parameters; others take byte arrays and structures. What do you mean, only 32-bit integers?" Well, behind the scenes, only 32-bit integers are being passed to the API functions. Although it looks like something else in your Visual Basic code, it does that to make things easier for you, which does in fact help most of the time.

Once again, ByRef literally means "By Reference." That "reference" is in reality a 32-bit pointer. That's right, Visual Basic is actually passing a 32-bit pointer to the object passed by reference. That structure isn't being passed -- only a pointer to it is. The same thing happens with anything else passed ByRef; that's why API functions can only edit your program variables when they are passed ByRef (except for strings). By using the pointer, the API function is able to access the variable itself to read or change.

The ByVal keyword is pretty much what it appears to be. It does pass the actual value of the parameter to the function. Again, that's why ByVal-passed variables (except for strings) cannot be edited by the function; the API function has no way to access the variable itself. API functions require ByVal whenever they do not need to modify the value of a 32-bit integer.

The Truth About Strings

You probably noticed that, in the above section, I kept making an exception for strings. While strings are always passed ByVal, the API function is always able to modify them freely. On the surface, it looks like this is impossible: nothing passed ByVal can be modified, right? But remember that every "real" parameter passed to an API function is a 32-bit integer. Strings passed ByVal aren't 32-bit integers, right?
It turns out that strings are much more sinister than you can imagine. It all stems from the differences between Visual Basic and C++. In C++, there is no “string” data type. Instead, an array of byte-long elements (of the "char"acter data type) is used to represent a string, each element in the array storing a single letter. The final element in the array is equal to 0, the null character. Whenever these pseudostrings are used, especially in parameter passing, a pointer to the first element is used. This pointer effectively references the entire string: everything from the pointed-at element to the terminating null.

Why does any of this matter? Remember that the Windows API is written in C++ and therefore uses the C++ style of string. Visual Basic is forced to use the Windows API to execute virtually all of its string operations (implicitly through Visual Basic’s intrinsic functions). This forces Visual Basic to internally use the C++ style strings all the time while providing the programmer with its own easier-to-use string data type.

In reality, the Visual Basic String data type is actually a special form of the Long data type! Throughout the entire Visual Basic command set, the programming language recognizes the difference between a String and a Long and acts accordingly. (For example, the Len function returns the number of bytes in the actual string, not the length of the “real” variable.) And what does this covert 32-bit integer hold? Naturally, it holds a pointer to the beginning of the actual string! This is why strings must always be passed ByVal: the string itself is not passed, but the pointer to it is. Similar reasoning explains why API functions can edit these strings despite having been passed ByVal.

Don't believe what I just said? I can prove it to you. Consider the following structure:

Public Type MYSTRUCT
    svar As String
End Type

This structure has a single data member: a variable-length string. Now ask yourself, what is the size of the structure? Does it depend on the content of the string? Or is it a constant value? Try running the following code and you'll discover the answer:

Dim st As MYSTRUCT
st.svar = "This is a line of text."
Debug.Print "Size of structure is"; Len(st)
st.svar = "Hello, world!"
Debug.Print "Size of structure is"; Len(st)

Run this code and you'll notice that both times, the size of the structure is reported to be 4. This is in fact the size in bytes of a 32-bit integer! The structure does not actually store a string. Instead, it stores a 32-bit pointer to the string. When you think about it, there's no other way to implement a variable-length string inside a structure. If the string truly were embedded in the structure, its entire contents would have to be rearranged every time the length of the string changed.

But what about fixed-length strings? It turns out that they are what they claim to be; they are not pointers. Try changing the data member in MYSTRUCT to the data type "String * 40" and run the example. Both times, the reported size will be 40. For fixed-length strings in a structure, the contents of the string are in fact embedded in the structure. However, this holds only for structures. If you define a fixed-length string outside a structure, it will still actually be a pointer to a string and must be passed ByVal to any API function.

Has all of this been confusing? It probably has, since you learned the truth about one of the great deceptions Visual Basic makes. This information really doesn't apply to anything else besides API function programming. Nevertheless, this realization should explain the curious use of ByVal for strings. Besides, this advanced information will give you a better understanding about what you're typing into your program.
The ByVal Keyword and CLng() Function

The ByVal method of parameter passing is often used when a pointer to some object is needed. However, in some cases, you wish to specify no object at all. For example, with some API functions, if you do not use an optional feature which requires a certain structure, passing that structure anyway will cause the function to fail. You need to somehow pass "nothing" for a ByVal parameter.

The way to accomplish this is to make use of the ByVal keyword. Placing the ByVal keyword immediately before the desired parameter, that parameter will be passed by value regardless of what appears in the function's declaration. Then you are able to simply pass 0 as the parameter, effectively giving a null pointer to the function. But what if that parameter is not a Long data type? Change the declaration to make that parameter's data type Any! But then, when you pass a null pointer, you must use the expression ByVal CLng(0).

Why "ByVal CLng(0)"? As the previous paragraph stated, ByVal is necessary to pass 0 for the parameter, instead of a pointer to where that 0 is stored in memory (and that pointer would not be 0). CLng() is a data type conversion function which converts a number into the Long data type. If you did not include the call to CLng(), Visual Basic would not know which data type the 0 is. Since the function declaration, reading As Any for that parameter, would not offer any help, it would probably try to pass it as a regular Integer or some other non-32-bit integer. But since API functions only accept 32-bit integers for parameters, a fatal error would arise. Therefore, using "ByVal CLng(0)" assures that a fully 32-bit 0 is being passed to the function, and everything will work perfectly.

For an example of ByVal CLng(0) in action, look at the following example:

```vba
' Read both a Long (32-bit) number and a String from the file
' C:\Test\myfile.txt. Notice how the ByVal keyword must be used
' when reading a string variable.
Dim longbuffer As Long  ' receives long read from file
Dim stringbuffer As String  ' receives string read from file
Dim numread As Long  ' receives number of bytes read from file
Dim hFile As Long  ' handle of the open file
Dim retval As Long  ' return value

' Open the file for read-level access.
hFile = CreateFile("C:\Test\myfile.txt", GENERAL_READ, FILE_SHARE_READ, ByVal CLng(0), OPEN_EXISTING, FILE_ATTRIBUTE_ARCHIVE, 0)
If hFile = -1 Then  ' the file could not be opened
    Debug.Print "Unable to open the file -- it probably does not exist."
    End  ' abort the program
End If

' Read a Long-type number from the file
retval = ReadFile(hFile, longbuffer, Len(longbuffer), numread, ByVal CLng(0))
If numread < Len(longbuffer) Then  ' EOF reached
    Debug.Print "End of file encountered -- could not read the data."
Else
    Debug.Print "Number read from file:”; longbuffer
End If

' Read a 10-character string from the file
```
stringbuffer = Space(10)  ' make room in the buffer
retval = ReadFile(hFile, ByVal stringbuffer, 10, numread, ByVal CLng(0))
If numread = 0 Then  ' EOF reached
    Debug.Print "End of file encountered -- could not read any data."
ElseIf numread < 10 Then  ' read between 0 and 10 bytes
    Debug.Print "Incomplete string read: "; Left(stringbuffer, numread)
Else
    Debug.Print "String read from file: "; stringbuffer
End If

' Close the file.
retval = CloseHandle(hFile)

Remember: if you ever have to pass a "null pointer" (i.e., a value of 0 for a ByVal) to a function, you must change that parameter's data type in the declaration to Any, in case it is something different. If that parameter is not ByVal according to the declaration, then the ByVal keyword must be used explicitly in the call to the function. Finally, you must make use of the CLng() conversion function to force the 0 to be fully 32 bits in length.

Getting Pointers to Other Objects

Sometimes, an API function will demand a pointer to some sort of object, in which case you must find a way to obtain the necessary pointer. Sadly, the AddressOf operator only works with application-defined functions, so it cannot provide a pointer to a structure or other object. So what can you do?

Actually, there is no way in Visual Basic to get a pointer to any object. Instead, you should create a memory block of the necessary length and copy the object (often a structure) into that block. By using the GlobalAlloc and GlobalLock functions, both a handle and a pointer to this memory block can be obtained. Then, when calling the necessary API function, you can simply use a pointer to the memory block instead of one to the original object. Since the memory block holds a copy of the data (generated via the lstrcpy function for strings, or the CopyMemory function for all other objects), there is effectively no difference between it and the original object. After using the block, it should be freed using the GlobalUnlock and GlobalFree functions as necessary.

The easiest way of learning how to create and manipulate these pure memory blocks is to examine an example of it in use. The following example opens the Choose Font common dialog box. The CHOOSEFONT_TYPE structure requires a pointer to a LOGFONT structure as one of its data members. This is done by using a memory block (referenced via hMem and pMem) to hold a copy of the structure. The structure's contents is initially copied into the memory block, and later the block is copied back to the structure in case any alterations were made to its contents.

' Display a Choose Font dialog box. Print out the typeface name, point size, ' and style of the selected font. More detail about topics in this example can be found in ' the pages for CHOOSEFONT_TYPE and LOGFONT.
Dim cf As CHOOSEFONT_TYPE  ' data structure needed for function
Dim lfont As LOGFONT  ' receives information about the chosen font
Dim hMem As Long, pMem As Long  ' handle and pointer to memory buffer
Dim fontname As String  ' receives name of font selected
Dim retval As Long  ' return value
Initialize the default selected font: Times New Roman, regular, black, 12 point.
(Note that some of that information is in the CHOOSEFONT_TYPE structure instead.)

lfont.lfHeight = 0 ' determine default height
lfont.lfWidth = 0 ' determine default width
lfont.lfEscapement = 0 ' angle between baseline and escapement vector
lfont.lfOrientation = 0 ' angle between baseline and orientation vector
lfont.lfWeight = FW_NORMAL ' normal weight i.e. not bold
lfont.lfItalic = 0 ' not italic
lfont.lfUnderline = 0 ' not underline
lfont.lfStrikeOut = 0 ' not strikeout
lfont.lfCharSet = DEFAULT_CHARSET ' use default character set
lfont.lfOutPrecision = OUT_DEFAULT_PRECIS ' default precision mapping
lfont.lfClipPrecision = CLIP_DEFAULT_PRECIS ' default clipping precision
lfont.lfQuality = DEFAULT_QUALITY ' default quality setting
lfont.lfPitchAndFamily = DEFAULT_PITCH Or FF_ROMAN ' default pitch, proportional with serifs
lfont.lfFaceName = "Times New Roman" & vbNullChar ' string must be null-terminated

Create the memory block which will act as the LOGFONT structure buffer.
hMem = GlobalAlloc(GMEM_MOVEABLE Or GMEM_ZEROINIT, Len(lfont))
pMem = GlobalLock(hMem)  ' lock and get pointer
CopyMemory ByVal pMem, lfont, Len(lfont)  ' copy structure's contents into block

' Initialize dialog box: Screen and printer fonts, point size between 10 and 72.
cf.lStructSize = Len(cf)  ' size of structure
cf.hwndOwner = Form1.hWnd ' window Form1 is opening this dialog box
cf.hdc = Printer.hDC  ' device context of default printer (using VB's mechanism)
cf.lfLogFont = pMem  ' pointer to LOGFONT memory block buffer
cf.iPointSize = 120 ' 12 point font (in units of 1/10 point)
cf.flags = CF_BOTH Or CF_EFFECTS Or CF_FORCEFONTEXIST Or CF_INITTOLOGFONTSTRUCT Or CF_LIMITSIZE

' Now, call the function. If successful, copy the LOGFONT structure back into the structure
' and then print out the attributes we mentioned earlier that the user selected.
retval = ChooseFont(cf)  ' open the dialog box
If retval <> 0 Then ' success
    CopyMemory lfont, ByVal pMem, Len(lfont)  ' copy memory back
    Now make the fixed-length string holding the font name into a "normal" string.
    fontname = Left(lfont.lfFaceName, InStr(lfont.lfFaceName, vbNullChar) - 1)
    ' Display font name and a few attributes.
    Debug.Print "FONT NAME: "; fontname
Debug.Print "FONT SIZE (points):"; cf.iPointSize / 10 ' in units of 1/10 point!
Debug.Print "FONT STYLE(S): ";
If lfont.lfWeight >= FW_BOLD Then Debug.Print "Bold ";
If lfont.lfItalic <> 0 Then Debug.Print "Italic ";
If lfont.lfUnderline <> 0 Then Debug.Print "Underline ";
If lfont.lfStrikeOut <> 0 Then Debug.Print "Strikeout";
Debug.Print ' end the line
End If

' Deallocate the memory block we created earlier. Note that this must ' be done whether the function succeeded or not.
retval = GlobalUnlock (hMem)  ' destroy pointer, unlock block
retval = GlobalFree (hMem)  ' free the allocated memory

<< Back to Part 5 | Contents of Introduction | (no following page)