Windows Libraries

From QB64.org wiki

QB64 can support the **specific Windows Operating System Libraries** on your PC. They should be located in the **System32** folder. Use DECLARE LIBRARY with the appropriate ALIAS. Loaded DLL files are NOT required to be named in the Declaration!

Note: C++ Header files should be placed in the QB64 folder and are not required after a program is compiled.
Note: QB64 requires all DLL files to either be with the program or in the C:\WINDOWS\SYSTEM32 folder!
Maximum Windows path: MAX_PATH = drive letter + ":\" + 256 + CHR$(0) = 260 characters.

Your code contribution using the Windows Libraries could end up here!

Windows API Data Structures

Name	Description	Bits	QB64 Type
bit	8 bits in one byte	1	_BIT
nybble	2 nybbles in one byte	4	_BIT * 4
byte	1 byte (2 nybbles)	8	_BYTE
Boolean	1 byte (signed/unsigned)	8	_BYTE
CharA(FunctA)	ASCII character	8(LEN(buffer))	_BYTE
WORD	2 bytes	16	INTEGER
CharW(FunctW)	Unicode wide character	16(LEN(buffer)\2)	INTEGER
DWORD	4 bytes	32	LONG
QWORD	8 bytes	64	_INTEGER64
LP or hwnd	Short or Long Pointer	ANY INTEGER	_OFFSET

Void * in C code is also an _OFFSET

Windows Data Types (http://msdn.microsoft.com/en-us/library/aa383751(v=vs.85).aspx)
Windows 32 API constant values (http://doc.pcsoft.fr/en-US/?6510001)

Contents

Computer Date/Time

```
'=============
'DATE&TIME.BAS
'=============
'A DATE$ and TIME$ alternative.
'Gets & Sets System DATE & TIME using Windows API.
'Coded for QB64 by Dav FEB/2012

'=========================================================================
'NOTE: THIS DEMO WILL ATTEMPT TO CHANGE YOUR SYSTEM DATE FOR 5 SECONDS.
'     AFTER 5 SECONDS IT WILL ATTEMPT TO RESTORE ORIGINAL DATE SETTING.
'=========================================================================

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

DECLARE DYNAMIC LIBRARY "Kernel32"
'== NOTE: SetSystemTime& returns Zero if it Fails.
'== NOTE: GetSystemTime does not return a value.
    FUNCTION SetSystemTime& (lpSystemTime AS SYSTEMTIME)
    SUB GetSystemTime (lpSystemTime AS SYSTEMTIME)
END DECLARE

' Holds current values...
DIM CurrentTime AS SYSTEMTIME

' For the new values to set...
DIM NewTime AS SYSTEMTIME

' Get & Show current System Date/Time
GetSystemTime CurrentTime

PRINT
PRINT "--------------------------------"
PRINT "Current System DATE & TIME is..."
PRINT "DATE:"; CurrentTime.wMonth; "-"; CurrentTime.wDay; "-"; CurrentTime.wYear
PRINT "TIME:"; CurrentTime.wHour; ":"; CurrentTime.wMinute; ":"; CurrentTime.wSecond
PRINT "--------------------------------"


'=== Now Set new DATE only
'=== NOTICE we're using CurrentTime TIME value's '<==
'=== So we're just changing the DATE, not the TIME here...

NewTime.wYear = 2011     'move date back to prevent trial version expiration!
```

```
NewTime.wMonth = 12
NewTime.wDayOfWeek = -1
NewTime.wDay = 25
NewTime.wHour = CurrentTime.wHour '<==
NewTime.wMinute = CurrentTime.wMinute '<==
NewTime.wSecond = CurrentTime.wSecond '<==
NewTime.wMilliseconds = CurrentTime.wMilliseconds '<==

' Set the new values
x = SetSystemTime&(NewTime)
IF x = 0 THEN PRINT "Failed to change DATE/TIME!": END

' Grab new System DATE settings...
' Temporary holding space...for TIME...
DIM CurrentTime2 AS SYSTEMTIME

GetSystemTime CurrentTime2

PRINT
PRINT "--------------------------------"
PRINT "Now the NEW System DATE is..."
PRINT "DATE:"; CurrentTime2.wMonth; "-"; CurrentTime2.wDay; "-"; CurrentTime2.wYear
PRINT "--------------------------------"
PRINT
PRINT "Waiting 5 seconds! Check computer date in taskbar....."


SLEEP 5

'=== Now set everything back to what it was at the beginning.
'=== Using TIME values from CurrentTime2 so we don't lose any
'=== seconds from SLEEPing....

' Grab current values again, keeps the running TIME...
GetSystemTime CurrentTime2

PRINT "Resetting DATE back..."

NewTime.wYear = CurrentTime.wYear
NewTime.wMonth = CurrentTime.wMonth
NewTime.wDayOfWeek = -1
NewTime.wDay = CurrentTime.wDay
NewTime.wHour = CurrentTime2.wHour '<==
NewTime.wMinute = CurrentTime2.wMinute '<==
NewTime.wSecond = CurrentTime2.wSecond '<==
NewTime.wMilliseconds = CurrentTime2.wMilliseconds '<==

' Set the DATE & TIME values back

x = SetSystemTime&(NewTime):
IF x = 0 THEN PRINT "Failed to change DATE/TIME!": END

' Now let's get & show Current values, see if it worked...

GetSystemTime CurrentTime

PRINT
PRINT "--------------------------------"
PRINT "Now the System DATE & TIME is..."
PRINT "DATE:"; CurrentTime.wMonth; "-"; CurrentTime.wDay; "-"; CurrentTime.wYear
PRINT "TIME:"; CurrentTime.wHour; ":"; CurrentTime.wMinute; ":"; CurrentTime.wSecond
PRINT "--------------------------------"

END
```

Code by Dav

(Return to Table of Contents)

Borderless Window

```
'============
'NOBORDER.BAS
'============

DECLARE CUSTOMTYPE LIBRARY
    FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$)
END DECLARE

DECLARE DYNAMIC LIBRARY "User32"
    FUNCTION GetWindowLongA& (BYVAL hwnd AS LONG, BYVAL nIndex AS LONG)
    FUNCTION SetWindowLongA& (BYVAL hwnd AS LONG, BYVAL nIndex AS LONG, BYVAL dwNewLong AS LONG)
    FUNCTION SetWindowPos& (BYVAL hwnd AS LONG, BYVAL hWndInsertAfter AS LONG, BYVAL x AS LONG, BYVAL y AS LONG
END DECLARE

GWL_STYLE = -16
WS_BORDER = &H800000

_TITLE "No Border"
hwnd& = FindWindow(0, "No Border" + CHR$(0))

PRINT "Press any key for no border...": A$ = INPUT$(1)
```

```
winstyle& = GetWindowLongA&(hwnd&, GWL_STYLE)
a& = SetWindowLongA&(hwnd&, GWL_STYLE, winstyle& AND NOT WS_BORDER)
a& = SetWindowPos&(hwnd&, 0, 0, 0, 0, 0, 39)

PRINT "Press any key to get back border...": SLEEP

winstyle& = GetWindowLongA&(hwnd&, GWL_STYLE)
a& = SetWindowLongA&(hwnd&, GWL_STYLE, winstyle& OR WS_BORDER)
a& = SetWindowPos&(hwnd&, 0, 0, 0, 0, 0, 39)

PRINT "The end"
```

Code by Dav

(Return to Table of Contents)

Color Dialog Box

The Color Dialog Box can set custom colors and alpha saturation levels.

```
' Color Dialog flag constants (use + or OR to use more than 1 flag)
CONST CC_RGBINIT = &H1& '           Sets the initial color (don't know how to set it)
CONST CC_FULLOPEN = &H2& '          Opens all dialog sections such as the custom color selector
CONST CC_PREVENTFULLOPEN = &H4& '   Prevents the user from opening the custom color selector
CONST CC_SHOWHELP = &H8& '          Shows the help button (USELESS!)
'-------------------------------------------------------------------------------------

TYPE COLORDIALOGTYPE
  lStructSize AS LONG '         Length of this TYPE structure
  hwndOwner AS LONG '           Dialog owner's handle
  hInstance AS LONG '           ?
  rgbResult AS LONG '           The RGB color the user selected
  lpCustColors AS _OFFSET '     Pointer to an array of 16 custom colors (will be changed by user)
  flags AS LONG '               Dialog flags
  lCustData AS LONG '           Custom data
  lpfnHook AS LONG '            Hook
  lpTemplateName AS _OFFSET '   Custom template
END TYPE

DIM ColorString AS STRING * 64
ColorString = "FFFFFFFFFF" 'not sure how this works?

DECLARE DYNAMIC LIBRARY "comdlg32"
  FUNCTION ChooseColorA& (DIALOGPARAMS AS COLORDIALOGTYPE) '    Yet the also famous color dialog box
END DECLARE

DECLARE LIBRARY
  FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$) ' To get hWnd handle
END DECLARE

SCREEN _NEWIMAGE(640, 480, 12) '32 or 16 or 256 color screen modes
_TITLE "Color Common Dialog demo" 'set Title of program
hWnd& = FindWindow(0, "Color Common Dialog demo" + CHR$(0)) 'get window handle using _TITLE string

clr~& = ChooseColor&(_RGB32(0, 0, 0), ColorString$, Cancel, CC_FULLOPEN, hWnd&)

CLS , clr~& 'make background chosen color
LOCATE 10, 31
IF Cancel <> -1 THEN
  COLOR _RGB(255, 255, 255) 'white text
  PRINT "Color:"; clr~&; "&H" + HEX$(clr~&) '  use last color chosen
  PRINT ColorString$ '              display user custom color values chosen
ELSE: PRINT "No color was chosen!"
END IF
END

FUNCTION ChooseColor& (InitialColor&, CustomColors$, Cancel, Flags&, hWnd&)
' Parameters:
'  InitialColor&  - The initial color used, will take effect if CC_RGBINIT flag is specified
'  CustomColors$  - A 64-byte string where the user's custom colors will be stored (4 bytes per color in RGB0 f
'  Cancel         - Variable where the cancel flag will be stored.
'  Flags&         - Dialog flags
'  hWnd&          - Your program's window handle that should be aquired by the FindWindow function.

DIM ColorCall AS COLORDIALOGTYPE

ColorCall.rgbResult = _RGB32(_BLUE32(InitialColor&), _GREEN32(InitialColor&), _RED32(InitialColor&))
ColorCall.lStructSize = LEN(ColorCall)
ColorCall.hwndOwner = hWnd&
ColorCall.flags = Flags&
ColorCall.lpCustColors = _OFFSET(CustomColors$)

' Do dialog call
Result = ChooseColorA(ColorCall)
IF Result THEN
  rgbResult& = ColorCall.rgbResult
  ' Swap RED and BLUE color intensity values using _RGB
  ChooseColor& = _RGB(_BLUE32(rgbResult&), _GREEN32(rgbResult&), _RED32(rgbResult&))
ELSE
  Cancel = -1
END IF
END FUNCTION
```

Adapted from code by Jobert14

Note: The ChooseColor value is converted using _RGB32 with the Blue and Red values being swapped.

Converting 32 bit Dialog Box Color values for 4 or 8 BPP Screen modes

```
SCREEN _NEWIMAGE(640, 480, 12) 'change from screen 12 to 32 to see the difference
rgbresult& = &H8080FF 'Dialog box long color reverse HEX$ return value
clr~& = _RGB(_BLUE32(rgbresult&), _GREEN32(rgbresult&), _RED32(rgbresult&)) 'swap red and blue
COLOR clr~&: PRINT clr~&, HEX$(clr~&) '_UNSIGNED LONG color values
```

Note: The _RGB value returned is full _ALPHA. Use _ALPHA or _RGBA to set the transparency in a program.

(Return to Table of Contents)

Desktop Size

Returns the Left, Top, Right and Bottom coordinates of the current desktop area.

```
DECLARE DYNAMIC LIBRARY "user32"
    FUNCTION SystemParametersInfoW& (BYVAL uiAction~&, BYVAL uiParam~&, BYVAL pvParam%&, BYVAL fWinlni~&)
END DECLARE

CONST SPI_GETWORKAREA = &H30

TYPE RECT
    left AS LONG
    top AS LONG
    right AS LONG
    bottom AS LONG
END TYPE
DIM Rec AS RECT

IF 0 = SystemParametersInfoW(SPI_GETWORKAREA, 0, _OFFSET(Rec), 0) THEN
    'function failed. You may call kernel32's GetLastError for more info.
    PRINT "failed."
END IF

PRINT Rec.left
PRINT Rec.top
PRINT Rec.right
PRINT Rec.bottom
PRINT
scr& = _SCREENIMAGE
PRINT _WIDTH(scr&)
PRINT _HEIGHT(scr&)
_FREEIMAGE scr&

END
```

(Return to Table of Contents)

Directory Environment

Returns various system environment settings including the current program's EXE name.

```
DECLARE LIBRARY   'Directory Information using KERNEL32 provided by Dav
FUNCTION WINDirectory ALIAS GetWindowsDirectoryA (lpBuffer AS STRING, BYVAL nSize AS LONG)
FUNCTION SYSDirectory ALIAS GetSystemDirectoryA (lpBuffer AS STRING, BYVAL nSize AS LONG)
FUNCTION CURDirectory ALIAS GetCurrentDirectoryA (BYVAL nBufferLen AS LONG, lpBuffer AS STRING)
FUNCTION TempPath ALIAS GetTempPathA (BYVAL nBufferLen AS LONG, lpBuffer AS STRING)
FUNCTION GetModuleFileNameA (BYVAL hModule AS LONG, lpFileName AS STRING, BYVAL nSize AS LONG)
END DECLARE

'=== SHOW WINDOWS DIRECTORY
WinDir$ = SPACE$(144)
Result = WINDirectory(WinDir$, LEN(WinDir$))
IF Result THEN PRINT "WINDOWS DIRECTORY: "; LEFT$(WinDir$, Result)

'=== SHOW SYSTEM DIRECTORY
SysDir$ = SPACE$(144)
Result = SYSDirectory(SysDir$, LEN(SysDir$))
IF Result THEN PRINT "SYSTEM DIRECTORY: "; LEFT$(SysDir$, Result)

'=== SHOW CURRENT DIRECTORY
CurDir$ = SPACE$(255)
Result = CURDirectory(LEN(CurDir$), CurDir$)
IF Result THEN PRINT "CURRENT DIRECTORY: "; LEFT$(CurDir$, Result)

'=== SHOW TEMP DIRECTORY
TempDir$ = SPACE$(100)
Result = TempPath(LEN(TempDir$), TempDir$)
IF Result THEN PRINT "TEMP DIRECTORY: "; LEFT$(TempDir$, Result)

'=== SHOW CURRENT PROGRAM
```

```
FileName$ = SPACE$(256)
Result = GetModuleFileNameA(0, FileName$, LEN(FileName$))
IF Result THEN PRINT "CURRENT PROGRAM: "; LEFT$(FileName$, Result)
END
```

Windows APIs courtesy of Dav

Returns the DOS 8.3 path and file name. The DLL used is in the QB64 folder.

```
DECLARE LIBRARY   'Directory Information using KERNEL32
FUNCTION GetShortPathNameA (lpLongPath AS STRING, lpShortPath AS STRING, BYVAL cBufferLen AS LONG)
END DECLARE

'=== SHOW SHORT PATH NAME
FileOrPath$ = "c:\qb64\SDL_image.dll"  '<< change to a relevant path or file name on computer
ShortPathName$ = SPACE$(260)
Result = GetShortPathNameA(FileOrPath$ + CHR$(0), ShortPathName$, LEN(ShortPathName$))
IF Result THEN PRINT "SHORT PATH NAME: " + ShortPathName$ ELSE PRINT "NOT Found!"
END
```

Courtesy of Dav

Disk Drives

Uses Kernel32 API to lists all available drives on system. Shows the drives type: HD/CD/DVD/RAM/NET/Removable/Unknown

```
CONST REMOVABLE = 2
CONST FIXED = 3
CONST REMOTE = 4
CONST CDROM = 5
CONST RAMDISK = 6

DECLARE LIBRARY
  FUNCTION GetDriveTypeA& (nDrive AS STRING)
  FUNCTION GetLogicalDriveStringsA (BYVAL nBuff AS LONG, lpbuff AS STRING)
END DECLARE

DIM DList AS STRING, DL AS STRING
DIM i AS LONG, typ AS LONG

i = GetLogicalDriveStringsA(0, DList) 'zero returns the drive string byte size
DList = SPACE$(i) 'set drive string length. Each drive is followed by CHR$(0)
i = GetLogicalDriveStringsA(i, DList) 'the byte size returns a string that long
PRINT DList

FOR n = 65 TO 90
  IF INSTR(DList, CHR$(n)) THEN
    DL = CHR$(n) + ":\" + CHR$(0)
    typ = GetDriveTypeA(DL)
    SELECT CASE typ
      CASE REMOVABLE: PRINT DL + "Removable"
      CASE FIXED: PRINT DL + "Fixed"
      CASE REMOTE: PRINT DL + "Remote"
      CASE CDROM: PRINT DL + "CDROM"
      CASE RAMDISK: PRINT DL + "RAM"
    END SELECT
  END IF
NEXT
```

Adapted from code by Dav

Note: The length of the string returned by *GetLogicalDriveStringsA* can be divided by 4 to tell the number of physical and ram drives.

(Return to Table of Contents)

File Attributes

```
DECLARE LIBRARY
    FUNCTION GetFileAttributes& (f$)
    FUNCTION SetFileAttributes& (f$, BYVAL attrib&)
END DECLARE

CONST INVALID_FILE_ATTRIBUTES = -1
CONST FILE_ATTRIBUTE_READONLY = 1
CONST FILE_ATTRIBUTE_HIDDEN = 2
CONST FILE_ATTRIBUTE_SYSTEM = 4
CONST FILE_ATTRIBUTE_DIRECTORY = 16
CONST FILE_ATTRIBUTE_ARCHIVE = 32

file$ = "temp.txt"

a = GetFileAttributes(file$)
PRINT a 'if no file, then you'll see a -1 here

OPEN file$ FOR OUTPUT AS #1
CLOSE #1

a = GetFileAttributes(file$)
```

```
PRINT a 'a new file, it prints 32 for me here

x = SetFileAttributes(file$, 1) 'set the read only flag
a = GetFileAttributes(file$)
PRINT a 'notice, it prints 1 here and not 32.  We didn't add a flag, we changed it completel
```

File Open and Save Dialog

Open and Save Dialog Boxes get file names

```
' Dialog flag constants (use + or OR to use more than 1 flag value)
CONST OFN_ALLOWMULTISELECT = &H200& '  Allows the user to select more than one file, not recommended!
CONST OFN_CREATEPROMPT = &H2000& '     Prompts if a file not found should be created(GetOpenFileName only).
CONST OFN_EXTENSIONDIFFERENT = &H400& 'Allows user to specify file extension other than default extension.
CONST OFN_FILEMUSTEXIST = &H1000& '    Chechs File name exists(GetOpenFileName only).
CONST OFN_HIDEREADONLY = &H4& '        Hides read-only checkbox(GetOpenFileName only)
CONST OFN_NOCHANGEDIR = &H8& '         Restores the current directory to original value if user changed
CONST OFN_NODEREFERENCELINKS = &H100000& 'Returns path and file name of selected shortcut(.LNK) file instead of
CONST OFN_NONETWORKBUTTON = &H20000& ' Hides and disables the Network button.
CONST OFN_NOREADONLYRETURN = &H8000& ' Prevents selection of read-only files, or files in read-only subdirector
CONST OFN_NOVALIDATE = &H100& '        Allows invalid file name characters.
CONST OFN_OVERWRITEPROMPT = &H2& '     Prompts if file already exists(GetSaveFileName only)
CONST OFN_PATHMUSTEXIST = &H800& '     Checks Path name exists (set with OFN_FILEMUSTEXIST).
CONST OFN_READONLY = &H1& '            Checks read-only checkbox. Returns if checkbox is checked
CONST OFN_SHAREAWARE = &H4000& '       Ignores sharing violations in networking
CONST OFN_SHOWHELP = &H10& '           Shows the help button (useless!)
'-------------------------------------------------------------------------------------------

DEFINT A-Z
TYPE FILEDIALOGTYPE
  lStructSize AS LONG '         For the DLL call
  hwndOwner AS LONG '           Dialog will hide behind window when not set correctly
  hInstance AS LONG '           Handle to a module that contains a dialog box template.
  lpstrFilter AS _OFFSET '      Pointer of the string of file filters
  lpstrCustFilter AS _OFFSET
  nMaxCustFilter AS LONG
  nFilterIndex AS LONG '        One based starting filter index to use when dialog is called
  lpstrFile AS _OFFSET '        String full of 0's for the selected file name
  nMaxFile AS LONG '            Maximum length of the string stuffed with 0's minus 1
  lpstrFileTitle AS _OFFSET '   Same as lpstrFile
  nMaxFileTitle AS LONG '       Same as nMaxFile
  lpstrInitialDir AS _OFFSET ' Starting directory
  lpstrTitle AS _OFFSET '       Dialog title
  flags AS LONG '               Dialog flags
  nFileOffset AS INTEGER '      Zero-based offset from path beginning to file name string pointed to by lpstrFil
  nFileExtension AS INTEGER '   Zero-based offset from path beginning to file extension string pointed to by lps
  lpstrDefExt AS _OFFSET '      Default/selected file extension
  lCustData AS LONG
  lpfnHook AS LONG
  lpTemplateName AS _OFFSET
END TYPE

DECLARE DYNAMIC LIBRARY "comdlg32"    ' Library declarations using _OFFSET types
  FUNCTION GetOpenFileNameA& (DIALOGPARAMS AS FILEDIALOGTYPE) ' The Open file dialog
  FUNCTION GetSaveFileNameA& (DIALOGPARAMS AS FILEDIALOGTYPE) ' The Save file dialog
END DECLARE

DECLARE LIBRARY
  FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$) ' To get hWnd handle
END DECLARE

_TITLE "FileOpen Common Dialog demo"                          'set Title of program
hWnd& = FindWindow(0, "Open and Save Dialog demo" + CHR$(0)) 'get window handle using _TITLE string

' Do the Open File dialog call!
Filter$ = "Batch files (*.bat)|*.BAT|JPEG images (*.jpg)|*.JPG|All files (*.*)|*.*"
Flags& = OFN_FILEMUSTEXIST + OFN_NOCHANGEDIR + OFN_READONLY  '    add flag constants here
OFile$ = GetOpenFileName$("YEAH! Common Dialogs in QB64!!!", ".\", Filter$, 1, Flags&, hWnd&)

IF OFile$ = "" THEN   ' Display Open dialog results
  PRINT "Shame on you! You didn't pick any file..."
ELSE
  PRINT "You picked this file: "
  PRINT OFile$
  IF (Flags& AND OFN_READONLY) THEN PRINT "Read-only checkbox checked." 'read-only value in return
END IF

_DELAY 5 ' Do the Save File dialog call!
Filter$ = "Basic files (*.bas)|*.BAS|All files (*.*)|*.*"
Flags& = OFN_OVERWRITEPROMPT + OFN_NOCHANGEDIR '   add flag constants here
SFile$ = GetSaveFileName$("Save will not create a file!!!", ".\", Filter$, 1, Flags&, hWnd&)

IF SFile$ = "" THEN     ' Display Save dialog results
  PRINT "You didn't save the file..."
ELSE
  PRINT "You saved this file: "
  PRINT SFile$
END IF
END

FUNCTION GetOpenFileName$ (Title$, InitialDir$, Filter$, FilterIndex, Flags&, hWnd&)
'  Title$      - The dialog title.
'  InitialDir$ - If this left blank, it will use the directory where the last opened file is
'  located. Specify ".\" if you want to always use the current directory.
'  Filter$     - File filters separated by pipes (|) in the same format as using VB6 common dialogs.
```

```
'  FilterIndex - The initial file filter to use. Will be altered by user during the call.
'  Flags&      - Dialog flags. Will be altered by the user during the call.
'  hWnd&       - Your program's window handle that should be aquired by the FindWindow function.
'
' Returns: Blank when cancel is clicked otherwise, the file name selected by the user.
' FilterIndex and Flags& will be changed depending on the user's selections.

DIM OpenCall AS FILEDIALOGTYPE  ' Needed for dialog call

fFilter$ = Filter$
FOR R = 1 TO LEN(fFilter$) ' Replace the pipes with character zero
  IF MID$(fFilter$, R, 1) = "|" THEN MID$(fFilter$, R, 1) = CHR$(0)
NEXT R
fFilter$ = fFilter$ + CHR$(0)

lpstrFile$ = STRING$(2048, 0) ' For the returned file name
lpstrDefExt$ = STRING$(10, 0) ' Extension will not be added when this is not specified
OpenCall.lStructSize = LEN(OpenCall)
OpenCall.hwndOwner = hWnd&
OpenCall.lpstrFilter = _OFFSET(fFilter$)
OpenCall.nFilterIndex = FilterIndex
OpenCall.lpstrFile = _OFFSET(lpstrFile$)
OpenCall.nMaxFile = LEN(lpstrFile$) - 1
OpenCall.lpstrFileTitle = OpenCall.lpstrFile
OpenCall.nMaxFileTitle = OpenCall.nMaxFile
OpenCall.lpstrInitialDir = _OFFSET(InitialDir$)
OpenCall.lpstrTitle = _OFFSET(Title$)
OpenCall.lpstrDefExt = _OFFSET(lpstrDefExt$)
OpenCall.flags = Flags&

Result = GetOpenFileNameA&(OpenCall) '            Do Open File dialog call!

IF Result THEN   ' Trim the remaining zeros
  GetOpenFileName$ = LEFT$(lpstrFile$, INSTR(lpstrFile$, CHR$(0)) - 1)
  Flags& = OpenCall.flags
  FilterIndex = OpenCall.nFilterIndex
END IF

END FUNCTION

FUNCTION GetSaveFileName$ (Title$, InitialDir$, Filter$, FilterIndex, Flags&, hWnd&)
'  Title$      - The dialog title.
'  InitialDir$ - If this left blank, it will use the directory where the last opened file is
'     located. Specify ".\" if you want to always use the current directory.
'  Filter$     - File filters separated by pipes (|) in the same format as VB6 common dialogs.
'  FilterIndex - The initial file filter to use. Will be altered by user during the call.
'  Flags&      - Dialog flags. Will be altered by the user during the call.
'  hWnd&       - Your program's window handle that should be aquired by the FindWindow function.

' Returns: Blank when cancel is clicked otherwise, the file name entered by the user.
' FilterIndex and Flags& will be changed depending on the user's selections.

DIM SaveCall AS FILEDIALOGTYPE  ' Needed for dialog call

fFilter$ = Filter$
FOR R = 1 TO LEN(fFilter$)  ' Replace the pipes with zeros
  IF MID$(fFilter$, R, 1) = "|" THEN MID$(fFilter$, R, 1) = CHR$(0)
NEXT R
fFilter$ = fFilter$ + CHR$(0)

lpstrFile$ = STRING$(2048, 0) ' For the returned file name
lpstrDefExt$ = STRING$(10, 0) ' Extension will not be added when this is not specified
SaveCall.lStructSize = LEN(SaveCall)
SaveCall.hwndOwner = hWnd&
SaveCall.lpstrFilter = _OFFSET(fFilter$)
SaveCall.nFilterIndex = FilterIndex
SaveCall.lpstrFile = _OFFSET(lpstrFile$)
SaveCall.nMaxFile = LEN(lpstrFile$) - 1
SaveCall.lpstrFileTitle = SaveCall.lpstrFile
SaveCall.nMaxFileTitle = SaveCall.nMaxFile
SaveCall.lpstrInitialDir = _OFFSET(InitialDir$)
SaveCall.lpstrTitle = _OFFSET(Title$)
SaveCall.lpstrDefExt = _OFFSET(lpstrDefExt$)
SaveCall.flags = Flags&

Result& = GetSaveFileNameA&(SaveCall) ' Do dialog call!

IF Result& THEN  ' Trim the remaining zeros
  GetSaveFileName$ = LEFT$(lpstrFile$, INSTR(lpstrFile$, CHR$(0)) - 1)
  Flags& = SaveCall.flags
  FilterIndex = SaveCall.nFilterIndex
END IF
END FUNCTION
```

Code courtesy of Jobert14

Note: The Open and Save Dialog boxes get user selections and do not actually open or create a file! Your program must do that.

Microsoft MSDN (http://msdn.microsoft.com/en-us/library/aa155724(v=office.10).aspx)

Common Dialog Flag Constants (http://msdn.microsoft.com/en-us/library/aa259317(v=vs.60).aspx)

In VB6, variable-length strings in user TYPEs are actually pointer _OFFSETs to those strings.

(Return to Table of Contents)

File Times

```
CONST GENERIC_READ = -&H80000000
CONST GENERIC_WRITE = &H40000000
CONST FILE_SHARE_READ = &H1
CONST FILE_SHARE_WRITE = &H2
CONST OPEN_EXISTING = &H3
CONST INVALID_HANDLE_VALUE = -1

DECLARE DYNAMIC LIBRARY "kernel32"
FUNCTION CreateFileA%& (BYVAL lpFileName AS _OFFSET, BYVAL dwDesiredAccess AS _UNSIGNED LONG, BYVAL dwShareMode
FUNCTION CloseHandle& (BYVAL hObject AS _OFFSET)
FUNCTION GetFileTime& (BYVAL hFile AS _OFFSET, BYVAL lpCreationTime AS _OFFSET, BYVAL lpLastAccessTime AS _OFFS
FUNCTION SetFileTime& (BYVAL hFile AS _OFFSET, BYVAL lpCreationTime AS _OFFSET, BYVAL lpLastAccessTime AS _OFFS
FUNCTION FileTimeToLocalFileTime& (BYVAL lpFileTime AS _OFFSET, BYVAL lpLocalFileTime AS _OFFSET)
FUNCTION LocalFileTimeToFileTime& (BYVAL lpLocalFileTime AS _OFFSET, BYVAL lpFileTime AS _OFFSET)
FUNCTION FileTimeToSystemTime& (BYVAL lpFileTime AS _OFFSET, BYVAL lpSystemTime AS _OFFSET)
FUNCTION SystemTimeToFileTime& (BYVAL lpSystemTime AS _OFFSET, BYVAL lpFileTime AS _OFFSET)
FUNCTION GetLastError& ()
END DECLARE

TYPE FILETIME
  dwLowDateTime AS _UNSIGNED LONG
  dwHighDateTime AS _UNSIGNED LONG
END TYPE

TYPE SYSTEMTIME
  wYear AS _UNSIGNED INTEGER
  wMonth AS _UNSIGNED INTEGER
  wDayOfWeek AS _UNSIGNED INTEGER
  wDay AS _UNSIGNED INTEGER
  wHour AS _UNSIGNED INTEGER
  wMinute AS _UNSIGNED INTEGER
  wSecond AS _UNSIGNED INTEGER
  wMilliseconds AS _UNSIGNED INTEGER
END TYPE

DIM CreateDate AS FILETIME
DIM ModifyDate AS FILETIME
DIM AccessDate AS FILETIME

DIM systime AS SYSTEMTIME

DIM FileName AS STRING
DIM FileHandle AS _OFFSET

FileName = "readme.txt" + CHR$(0) '<<<<<< Existing file in QB64 folder. Use existing file path!

FileHandle = CreateFileA%&(_OFFSET(FileName), GENERIC_READ, FILE_SHARE_READ OR FILE_SHARE_WRITE, 0, OPEN_EXISTI
IF FileHandle <> INVALID_HANDLE_VALUE THEN
  IF GetFileTime&(FileHandle, _OFFSET(CreateDate), _OFFSET(ModifyDate), _OFFSET(AccessDate)) THEN
    PRINT HEX$(CreateDate.dwLowDateTime) + HEX$(CreateDate.dwHighDateTime)
    PRINT HEX$(ModifyDate.dwLowDateTime) + HEX$(ModifyDate.dwHighDateTime)
    PRINT HEX$(AccessDate.dwLowDateTime) + HEX$(AccessDate.dwHighDateTime)
    PRINT
    IF FileTimeToSystemTime&(_OFFSET(CreateDate), _OFFSET(systime)) THEN
      PRINT "Creation time, in GMT, in decimal:"
      PRINT "Year:"; systime.wYear
      PRINT "Month:"; systime.wMonth, "("; MID$("JanFebMarAprMayJunJulAugSepOctNovDec", (systime.wMonth * 3) -
      PRINT "DayOfWeek:"; systime.wDayOfWeek, "("; MID$("SunMonTueWedThuFriSat", (systime.wDayOfWeek * 3) + 1,
      PRINT "Day"; systime.wDay
      PRINT "Hour"; systime.wHour
      PRINT "Minute"; systime.wMinute
      PRINT "Second"; systime.wSecond
      PRINT "Milliseconds"; systime.wMilliseconds
    ELSE
      PRINT "FileTimeToSystemTime failed. Error: 0x" + LCASE$(HEX$(GetLastError&))
    END IF
  ELSE
    PRINT "GetFileTime failed. Error: 0x" + LCASE$(HEX$(GetLastError&))
  END IF
  IF CloseHandle&(FileHandle) = 0 THEN
    PRINT "CloseHandle failed. Error: 0x" + LCASE$(HEX$(GetLastError&))
    END
  END IF
ELSE
  PRINT "CreateFileA failed. Error: 0x" + LCASE$(HEX$(GetLastError&))
  END
END IF
END
```

Code courtesy of Michael Calkins
Use your own existing file name and path in this procedure.

(Return to Table of Contents)

Focus

Sets Focus on program with *SetForegroundWindow* after maximizing a minimized program when Shift+A is pressed. (See Top Most Window)

```
'Uses GetKeyState Win API to monitor a Key state.
'This demo will maximize the window and focus on program when Shift+A is pressed.

DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION FindWindowA%& (BYVAL ClassName AS _OFFSET, WindowName$) 'find process handle by title
  FUNCTION GetKeyState% (BYVAL nVirtKey AS LONG) 'Windows virtual key presses
  FUNCTION ShowWindow& (BYVAL hwnd AS _OFFSET, BYVAL nCmdShow AS LONG) 'maximize process
  FUNCTION GetForegroundWindow%& 'find currently focused process handle
  FUNCTION SetForegroundWindow& (BYVAL hwnd AS _OFFSET) 'set foreground window process(focus)
END DECLARE

title$ = "Cheapo Hotkey (Shift+A)" 'title of program window
_TITLE title$ 'set program title
hwnd%& = FindWindowA(0, title$ + CHR$(0)) 'find this program's process handle

PRINT "Minimize window, click another then Press Shift+A to bring it back up."

'=== below minimizes it for you
_DELAY 4
x& = ShowWindow&(hwnd%&, 2)
'============================

DO
  IF GetKeyState(16) < 0 AND GetKeyState(ASC("A")) < 0 THEN '<==== Shift+A
    FGwin%& = GetForegroundWindow%& 'get current process in focus
    PRINT "Program Handle:"; hwnd%&; "Focus handle:"; FGwin%&

    y& = ShowWindow&(hwnd%&, 1) 'maximize minimized program

    IF FGwin%& <> hwnd%& THEN z& = SetForegroundWindow&(hwnd%&) 'set focus when necessary

    PRINT "That is all. Return values:"; x&; y&; z&

    'PUT PROGRAM CODE OR SUB CALLS HERE!
    '=== below minimizes it for you again after code is done
    '_DELAY 4
    'x& = ShowWindow&(hwnd%&, 2)
    '============================
    _DELAY 5: END 'delay allows user to not minimize the window
  END IF

  _LIMIT 30 'save CPU usage while waiting for key press

LOOP
```

Adapted by Ted Weissgerber from code by Dav

The *GetForegroundWindow* function finds the process currently in focus. See: Hot Keys for Windows Virtual Keys

Always brings unfocused or minimized program to the top with focus when Shift+A hotkey combination is pressed.

```
'===========
'HOTKEYS.BAS
'===========
'Windows Hotkey example.
'This demo sets Shift+A hotkey to maximize the program window when minimized.
'Returns focus to the program.
'Coded by Dav JULY/2012


DECLARE CUSTOMTYPE LIBRARY
  FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$)
  FUNCTION ShowWindow& (BYVAL hwnd AS LONG, BYVAL nCmdShow AS LONG)
END DECLARE
DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION SendMessageA& (BYVAL hwnd AS LONG, BYVAL wMsg AS LONG, BYVAL wParam AS LONG, BYVAL lParam AS LONG)
  FUNCTION DefWindowProcA& (BYVAL hwnd AS LONG, BYVAL wMsg AS LONG, BYVAL wParam AS LONG, BYVAL lParam AS LONG)
END DECLARE

_TITLE "My Focus Program"
hwnd = FindWindow(0, "My Focus Program" + CHR$(0)) 'get Windows program ID by title

CONST WM_SETHOTKEY = &H32
CONST WM_SHOWWINDOW = &H18
'SendMessage function key combination values
CONST HK_SHIFT = &H100 'NOTE: Send Message values are different than HotKey
CONST HK_CTRL = &H200  'All other keys are the same as the HotKey values.
CONST HK_ALT = &H400

PRINT "Press Shift+A to maximize or bring this window to top..."
_DELAY 3

'=== Tell windows what hotkey you want to use.
hot& = SendMessageA&(hwnd, WM_SETHOTKEY, HK_SHIFT + ASC("A"), 0) 'use uppercase second key
PRINT hot&;
'=== See if hotkey set ok...
IF hot& <> 1 THEN
  PRINT "Hotkey not set." + "Error: "; hot&
  END
END IF

'===  minimize program with this
'a& = ShowWindow&(hwnd, 6)

'=== Below tells Windows what to do when hotkey is pressed.
```

```
'===(it maximizes the program window and returns focus over and over).
top& = DefWindowProcA&(hwnd, WM_SHOWWINDOW, 0, 0)

DO: _LIMIT 1
  'do your program stuff here....
  PRINT hot&; top&;
LOOP UNTIL INKEY$ = CHR$(27)
END
```

Adapted from code by Dav

Note: The *SetHotKey* message tells the designated Windows program ID when to do *DefWondowProcA* to focus the window.

This can also focus on other program IDs! See also: Hot Keys for Windows Virtual Keys

Note: Minimized programs will always lose focus when minimized unless clicked in taskbar.

(Return to Table of Contents)

Folder Dialog Box

The SHBrowseForFolder function receives information about the folder selected by the user in Windows XP to 7.

```
DECLARE CUSTOMTYPE LIBRARY
    FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$)
END DECLARE

_TITLE "Super Window"
hwnd& = FindWindow(0, "Super Window" + CHR$(0))

TYPE BROWSEINFO  'typedef struct _browseinfo 'Microsoft MSDN
  hwndOwner AS LONG '              '  HWND
  pidlRoot AS _OFFSET '             '  PCIDLIST_ABSOLUTE
  pszDisplayName AS _OFFSET '      '  LPTSTR
  lpszTitle AS _OFFSET '           '  LPCTSTR
  ulFlags AS _UNSIGNED LONG        '  UINT
  lpfn AS _OFFSET '                '  BFFCALLBACK
  lParam AS _OFFSET '              '  LPARAM
  iImage AS LONG '                 '  int
END TYPE  'BROWSEINFO, *PBROWSEINFO, *LPBROWSEINFO;

DECLARE DYNAMIC LIBRARY "shell32"
  FUNCTION SHBrowseForFolder%& (x AS BROWSEINFO) 'Microsoft MSDN
  SUB SHGetPathFromIDList (BYVAL lpItem AS _OFFSET, BYVAL szDir AS _OFFSET) 'Microsoft MSDN
END DECLARE

DIM b AS BROWSEINFO
b.hwndOwner = hwnd
DIM s AS STRING * 1024
b.pszDisplayName = _OFFSET(s$)
a$ = "Choose a folder!!!" + CHR$(0)
b.lpszTitle = _OFFSET(a$)
DIM o AS _OFFSET
o = SHBrowseForFolder(b)
IF o THEN
    PRINT LEFT$(s$, INSTR(s$, CHR$(0)) - 1)
    DIM s2 AS STRING * 1024
    SHGetPathFromIDList o, _OFFSET(s2$)
    PRINT LEFT$(s2$, INSTR(s2$, CHR$(0)) - 1)
ELSE
    PRINT "Cancel?"
END IF
```

Code by Galleon

(Return to Table of Contents)

Font Dialog Box

This dialog box does not return the actual font file name. Refer to the Registered Fonts File procedure below this one.

```
' Constants assigned to Flags. A LONG numerical suffix defines those constants as LONG
CONST CF_APPLY = &H200& '              Displays Apply button
CONST CF_ANSIONLY = &H400& '           list ANSI fonts only
CONST CF_BOTH = &H3& '                 list both Screen and Printer fonts
CONST CF_EFFECTS = &H100&  '           Display Underline and Strike Through boxes
CONST CF_ENABLEHOOK = &H8& '           set hook to custom template
CONST CF_ENABLETEMPLATE = &H10& '      enable custom template
CONST CF_ENABLETEMPLATEHANDLE = &H20&
CONST CF_FIXEDPITCHONLY = &H4000&   '  list only fixed-pitch fonts
CONST CF_FORCEFONTEXIST = &H10000& '   indicate error when font not listed is chosen
CONST CF_INACTIVEFONTS = &H2000000& ' display hidden fonts in Win 7 only
CONST CF_INITTOLOGFONTSTRUCT = &H40& 'use the structure pointed to by the lpLogFont member
CONST CF_LIMITSIZE = &H2000& '         select font sizes only within nSizeMin and nSizeMax members
CONST CF_NOOEMFONTS = &H800& '         should not allow vector font selections
CONST CF_NOFACESEL = &H80000& '        prevent displaying initial selection in font name combo box.
CONST CF_NOSCRIPTSEL = &H800000& '     Disables the Script combo box
```

```
CONST CF_NOSIMULATIONS = &H1000& '    Disables selection of font simulations
CONST CF_NOSIZESEL = &H200000&  '     Disables Point Size selection
CONST CF_NOSTYLESEL = &H100000& '     Disables Style selection
CONST CF_NOVECTORFONTS = &H800&
CONST CF_NOVERTFONTS = &H1000000&
CONST CF_OEMTEXT = &H7&
CONST CF_PRINTERFONTS = &H2& '        list fonts only supported by printer associated with the device
CONST CF_SCALABLEONLY = &H20000& '    select only vector fonts, scalable printer fonts, and TrueType fonts
CONST CF_SCREENFONTS = &H1&  '        lists only the screen fonts supported by system
CONST CF_SCRIPTSONLY = &H400& '       lists all non-OEM, Symbol and ANSI sets only
CONST CF_SELECTSCRIPT = &H400000&  '  can only use set specified in the Scripts combo box
CONST CF_SHOWHELP = &H4&   '          displays Help button reference
CONST CF_TTONLY = &H40000&  '         True Type only
CONST CF_USESTYLE = &H80& '           copies style data for the user's selection to lpszStyle buffer
CONST CF_WYSIWYG = &H8000& '          only list fonts available on both the printer and display
' Font Types returned by nFontType
CONST BOLD_FONTTYPE = &H100&
CONST ITALIC_FONTTYPE = &H200&
CONST PRINTER_FONTTYPE = &H4000&
CONST REGULAR_FONTTYPE = &H400&
CONST SCREEN_FONTTYPE = &H2000&
CONST SIMULATED_FONTTYPE = &H8000&
' Font Weights assigned to lfWeight
CONST FW_DONTCARE = 0
CONST FW_THIN = 100
CONST FW_ULTRALIGHT = 200
CONST FW_LIGHT = 300
CONST FW_REGULAR = 400
CONST FW_MEDIUM = 500
CONST FW_SEMIBOLD = 600
CONST FW_BOLD = 700
CONST FW_ULTRABOLD = 800
CONST FW_HEAVY = 900

CONST DEFAULT_CHARSET = 1
CONST LF_DEFAULT = 0
CONST FF_ROMAN = 16
CONST LF_FACESIZE = 32
CONST GMEM_MOVEABLE = &H2
CONST GMEM_ZEROINIT = &H40
'-----------------------------------------------------------------------------------------------

DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION FindWindowA%& (BYVAL ClassName AS _OFFSET, BYVAL WindowName AS _OFFSET)
END DECLARE

DECLARE DYNAMIC LIBRARY "comdlg32"
  FUNCTION ChooseFontA& (BYVAL lpcf AS _OFFSET)
  FUNCTION CommDlgExtendedError& () '               'dialog box error checking procedure
END DECLARE

TYPE CHOOSEFONT
  lStructSize AS _UNSIGNED LONG
  hwndOwner AS _OFFSET
  HDC AS _OFFSET
  lpLogFont AS _OFFSET
  iPointSize AS LONG
  Flags AS LONG
  rgbColors AS _UNSIGNED LONG
  lCustData AS _OFFSET
  lpfnHook AS _OFFSET
  lpTemplateName AS _OFFSET
  hInstance AS _OFFSET
  lpszStyle AS _OFFSET
  nFontType AS LONG '  if used as Unsigned Integer add Integer padder below
  'padder AS INTEGER ' use only when nFontType is designated as Unsigned Integer
  nSizeMin AS LONG
  nSizeMax AS LONG
END TYPE

TYPE LOGFONT
  lfHeight AS LONG
  lfWidth AS LONG
  lfEscapement AS LONG
  lfOrientation AS LONG
  lfWeight AS LONG
  lfItalic AS _BYTE '    not 0 when user selected
  lfUnderline AS _BYTE ' not 0 when user selected
  lfStrikeOut AS _BYTE ' not 0 when user selected
  lfCharSet AS _BYTE
  lfOutPrecision AS _BYTE
  lfClipPrecision AS _BYTE
  lfQuality AS _BYTE
  lfPitchAndFamily AS _BYTE
  lfFaceName AS STRING * 32 'contains name listed in dialog
END TYPE

DIM hWnd AS _OFFSET 'must DIM or hWnd won't work
DIM Title AS STRING 'keeps Dialog with program window

SCREEN _NEWIMAGE(640, 480, 12) '32 bit, 16 or 256 color screen modes
Title$ = "Choose Font Dialog"
_TITLE Title$ 'set Title of program
_DELAY 1
hWnd = FindWindowA%&(0, _OFFSET(Title)) 'get window handle

Font$ = ShowFont$(hWnd) '           call Dialog Box and get the font selection
PRINT Font$, HEX$(FontColor&), FontType$; FontEff$, PointSize& 'other values SHARED by function
COLOR FontColor&: PRINT "Font Color Selected"
END
```

```
FUNCTION ShowFont$ (hWnd AS _OFFSET)
DIM cf AS CHOOSEFONT
DIM lfont AS LOGFONT
SHARED FontColor&, FontType$, FontEff$, PointSize AS LONG 'shared with main program
lfont.lfHeight = LF_DEFAULT ' determine default height '       set dailog box defaults
lfont.lfWidth = LF_DEFAULT ' determine default width
lfont.lfEscapement = LF_DEFAULT ' angle between baseline and escapement vector
lfont.lfOrientation = LF_DEFAULT ' angle between baseline and orientation vector
lfont.lfWeight = FW_REGULAR ' normal weight i.e. not bold
lfont.lfCharSet = DEFAULT_CHARSET ' use default character set
lfont.lfOutPrecision = LF_DEFAULT ' default precision mapping
lfont.lfClipPrecision = LF_DEFAULT ' default clipping precision
lfont.lfQuality = LF_DEFAULT ' default quality setting
lfont.lfPitchAndFamily = LF_DEFAULT OR FF_ROMAN ' default pitch, proportional with serifs
lfont.lfFaceName = "Times New Roman" + CHR$(0) ' string must be null-terminated
cf.lStructSize = LEN(cf) ' size of structure
cf.hwndOwner = hWnd ' window opening the dialog box
'cf.HDC = Printer.hDC ' device context of default printer (using VB's mechanism)
cf.lpLogFont = _OFFSET(lfont)
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.nFontType = REGULAR_FONTTYPE ' regular font type i.e. not bold or anything
cf.nSizeMin = 10 ' minimum point size
cf.nSizeMax = 72 ' maximum point size

IF ChooseFontA&(_OFFSET(cf)) <> 0 THEN '    'Initiate Dialog and Read user selections
  ShowFont = LEFT$(lfont.lfFaceName, INSTR(lfont.lfFaceName, CHR$(0)) - 1)
  'returns closest color attribute or 32 bit value and swaps red and blue color values
  FontColor& = _RGB(_BLUE32(cf.rgbColors), _GREEN32(cf.rgbColors), _RED32(cf.rgbColors))
  IF cf.nFontType AND BOLD_FONTTYPE THEN FontType$ = "Bold"
  IF cf.nFontType AND ITALIC_FONTTYPE THEN FontType$ = FontType$ + "Italic"
  IF cf.nFontType AND REGULAR_FONTTYPE THEN FontType$ = "Regular"
  IF lfont.lfUnderline THEN FontEff$ = "Underline"
  IF lfont.lfStrikeOut THEN FontEff$ = FontStyle$ + "Strikeout"
  PointSize = cf.iPointSize \ 10
ELSE
  IF CommDlgExtendedError& THEN
    PRINT "ChooseFontA failed. Error: 0x"; LCASE$(HEX$(CommDlgExtendedError&))
  ELSE: PRINT "Entry was cancelled!"
  END IF
END IF
END FUNCTION
```

Code by Michael Calkins and Ted Weissgerber

Warning! This dialog box may error for no apparent reason! See CommDlgExtendedError (http://msdn.microsoft.com/en-us/library/windows/desktop/ms646916(v=vs.85).aspx) for more info!

Snippet: Shows how to compare a Font Dialog Box request with STRING file data created by the Registry Font Library below:

```
' procedure assumes that all fonts have been loaded into an array as below:
RegFont$(1) = "Times New Roman (TrueType) = TIMES.TTF" 'array simulates registry data from file
RegFont$(2) = "Times New Roman Bold (TrueType) = TIMESBD.TTF"
RegFont$(3) = "Times New Roman Bold Italic (TrueType) = TIMESBI.TTF"
RegFont$(4) = "Times New Roman Italic (TrueType) = TIMESI.TTF"
Font$ = "Times New Roman Bold Italic" 'font name returned by Font Dialog box
File$ = ""

FOR n = 1 TO 4 'numFiles% 'actual number of font file registry records
IF INSTR$(RegFont$(n), Font$) THEN 'check for match of Dialog Box font name "Times New Roman"
  FontFile$ = MID$(RegFont$(n), INSTR(RegFont$(n), "=") + 2)   'get each file name value
  SELECT CASE FontType$   'check for user requested font type to get file name
  CASE "Bold"
    IF INSTR(RegFont$(n), "Bold") AND INSTR(RegFont$(n), "Italic") = 0 THEN File$ = FontFile$
  CASE "Italic"
    IF INSTR(RegFont$(n), "Italic") AND INSTR(RegFont$(n), "Bold") = 0 THEN File$ = FontFile$
  CASE "BoldItalic"
    IF INSTR(RegFont$(n), "Bold") AND INSTR(RegFont$(n), "Italic") THEN File$ = FontFile$
  CASE ELSE 'regular font as default
    IF INSTR(RegFont$(n), "Bold") = 0 AND INSTR(RegFont$(n), "Italic") = 0 THEN File$ = FontFile$
  END SELECT
END IF
IF LEN(File$) THEN EXIT FOR  'quit searching
NEXT
```

Note: The Font Dialog name will not normally have descriptions such as Regular, Bold, Italics or (TrueType) so it ignores them.

GDI Tools - Font Guidlines (http://www.functionx.com/win32/Lesson14.htm)

(Return to Table of Contents)

Registered Fonts

The Registry lists the Font Names and associated TTF file names that are needed with the Font Dialog Box in a program.
The following program uses Registry functions from *advapi32.dll* to read the list of registered fonts and put them into a file.

```
' winreg.h
CONST HKEY_CLASSES_ROOT = &H80000000~&
CONST HKEY_CURRENT_USER = &H80000001~&
```

```
CONST HKEY_LOCAL_MACHINE = &H80000002~&
CONST HKEY_USERS = &H80000003~&
CONST HKEY_PERFORMANCE_DATA = &H80000004~&
CONST HKEY_CURRENT_CONFIG = &H80000005~&
CONST HKEY_DYN_DATA = &H80000006~&
CONST REG_OPTION_VOLATILE = 1
CONST REG_OPTION_NON_VOLATILE = 0
CONST REG_CREATED_NEW_KEY = 1
CONST REG_OPENED_EXISTING_KEY = 2

' http://msdn.microsoft.com/en-us/library/ms724884(v=VS.85).aspx
CONST REG_NONE = 0
CONST REG_SZ = 1
CONST REG_EXPAND_SZ = 2
CONST REG_BINARY = 3
CONST REG_DWORD_LITTLE_ENDIAN = 4  '   value is defined REG_DWORD in Windows header files
CONST REG_DWORD = 4 '                  32-bit number
CONST REG_DWORD_BIG_ENDIAN = 5 '       some UNIX systems support big-endian architectures
CONST REG_LINK = 6
CONST REG_MULTI_SZ = 7
CONST REG_RESOURCE_LIST = 8
CONST REG_FULL_RESOURCE_DESCRIPTOR = 9
CONST REG_RESOURCE_REQUIREMENTS_LIST = 10
CONST REG_QWORD_LITTLE_ENDIAN = 11  '  64-bit number in little-endian format
CONST REG_QWORD = 11 '                 64-bit number
CONST REG_NOTIFY_CHANGE_NAME = 1
CONST REG_NOTIFY_CHANGE_ATTRIBUTES = 2
CONST REG_NOTIFY_CHANGE_LAST_SET = 4
CONST REG_NOTIFY_CHANGE_SECURITY = 8

' http://msdn.microsoft.com/en-us/library/ms724878(v=VS.85).aspx
CONST KEY_ALL_ACCESS = &HF003F&
CONST KEY_CREATE_LINK = &H0020&
CONST KEY_CREATE_SUB_KEY = &H0004&
CONST KEY_ENUMERATE_SUB_KEYS = &H0008&
CONST KEY_EXECUTE = &H20019&
CONST KEY_NOTIFY = &H0010&
CONST KEY_QUERY_VALUE = &H0001&
CONST KEY_READ = &H20019&
CONST KEY_SET_VALUE = &H0002&
CONST KEY_WOW64_32KEY = &H0200&
CONST KEY_WOW64_64KEY = &H0100&
CONST KEY_WRITE = &H20006&

' winerror.h
' http://msdn.microsoft.com/en-us/library/ms681382(v=VS.85).aspx
CONST ERROR_SUCCESS = 0
CONST ERROR_FILE_NOT_FOUND = &H2&
CONST ERROR_INVALID_HANDLE = &H6&
CONST ERROR_MORE_DATA = &HEA&
CONST ERROR_NO_MORE_ITEMS = &H103&
'-----------------------------------------------------------------------------------------------
' REGSAM is an ACCESS_MASK (winreg.h), which is a DWORD (winnt.h)

'Note: All of these functions, except RegCloseKey, have both ANSI (ending in A)
'and Unicode (ending in W) versions. I am not aware of any reason why both
'versions could not be used in the same program. To add the Unicode version,
'duplicate the function declaration, but change the ending A to W. Be sure that
'you know how to use the Unicode version! ANSI versions tested, sort of:

DECLARE DYNAMIC LIBRARY "advapi32"

  ' http://msdn.microsoft.com/en-us/library/ms724897(v=VS.85).aspx
  FUNCTION RegOpenKeyExA& (BYVAL hKey AS _OFFSET, BYVAL lpSubKey AS _OFFSET, BYVAL ulOptions AS _UNSIGNED LONG,

  ' http://msdn.microsoft.com/en-us/library/ms724837(v=VS.85).aspx
  FUNCTION RegCloseKey& (BYVAL hKey AS _OFFSET)

  ' http://msdn.microsoft.com/en-us/library/ms724865(v=VS.85).aspx
  FUNCTION RegEnumValueA& (BYVAL hKey AS _OFFSET, BYVAL dwIndex AS _UNSIGNED LONG, BYVAL lpValueName AS _OFFSET

END DECLARE

DIM hKey AS _OFFSET
DIM Ky AS _OFFSET
DIM SubKey AS STRING
DIM Value AS STRING
DIM bData AS STRING
DIM t AS STRING
DIM dwType AS _UNSIGNED LONG
DIM numBytes AS _UNSIGNED LONG
DIM numTchars AS _UNSIGNED LONG
DIM l AS LONG
DIM dwIndex AS _UNSIGNED LONG

OPEN "F0NTList.INF" FOR OUTPUT AS #1  'create a new file for font data
PRINT
PRINT "This key lists the registered fonts available to all users:"
Ky = HKEY_LOCAL_MACHINE
SubKey = "SOFTWARE\Microsoft\Windows NT\CurrentVersion\Fonts" + CHR$(0)
Value = SPACE$(261) 'ANSI Value name limit 260 chars + 1 null
bData = SPACE$(&H7FFF) 'arbitrary

l = RegOpenKeyExA(Ky, _OFFSET(SubKey), 0, KEY_READ, _OFFSET(hKey))
IF l THEN
  PRINT "RegOpenKeyExA failed. Error: 0x" + LCASE$(HEX$(l))
ELSE
  PRINT whatKey$(Ky) + "\" + SubKey
  dwIndex = 0
  DO
    _DELAY .1
```

```
    numBytes = LEN(bData)
    numTchars = LEN(Value)
    l = RegEnumValueA(hKey, dwIndex, _OFFSET(Value), _OFFSET(numTchars), 0, _OFFSET(dwType), _OFFSET(bData), _(
    IF l THEN
      IF l <> ERROR_NO_MORE_ITEMS THEN
        PRINT "RegEnumValueA failed. Error: 0x" + LCASE$(HEX$(l))
      END IF
      EXIT DO
    ELSE
      PRINT whatType(dwType) + " " + LEFT$(Value, numTchars) + " = " + formatData(dwType, numBytes, bData)
      PRINT #1, LEFT$(Value, numTchars) + " = " + formatData(dwType, numBytes, bData)
    END IF
    dwIndex = dwIndex + 1
  LOOP
  CLOSE #1
  PRINT dwIndex; "Values."
  l = RegCloseKey(hKey)
  IF l THEN
    PRINT "RegCloseKey failed. Error: 0x" + LCASE$(HEX$(l))
    END
  END IF
END IF

END

FUNCTION whatType$ (dwType AS _UNSIGNED LONG)
SELECT CASE dwType
  CASE REG_SZ: whatType = "REG_SZ"
  CASE REG_EXPAND_SZ: whatType = "REG_EXPAND_SZ"
  CASE REG_BINARY: whatType = "REG_BINARY"
  CASE REG_DWORD: whatType = "REG_DWORD"
  CASE REG_DWORD_BIG_ENDIAN: whatType = "REG_DWORD_BIG_ENDIAN"
  CASE REG_LINK: whatType = "REG_LINK"
  CASE REG_MULTI_SZ: whatType = "REG_MULTI_SZ"
  CASE REG_RESOURCE_LIST: whatType = "REG_RESOURCE_LIST"
  CASE REG_FULL_RESOURCE_DESCRIPTOR: whatType = "REG_FULL_RESOURCE_DESCRIPTOR"
  CASE REG_RESOURCE_REQUIREMENTS_LIST: whatType = "REG_RESOURCE_REQUIREMENTS_LIST"
  CASE REG_QWORD: whatType = "REG_QWORD"
  CASE ELSE: whatType = "unknown"
END SELECT
END FUNCTION

FUNCTION whatKey$ (hKey AS _OFFSET)
SELECT CASE hKey
  CASE HKEY_CLASSES_ROOT: whatKey = "HKEY_CLASSES_ROOT"
  CASE HKEY_CURRENT_USER: whatKey = "HKEY_CURRENT_USER"
  CASE HKEY_LOCAL_MACHINE: whatKey = "HKEY_LOCAL_MACHINE"
  CASE HKEY_USERS: whatKey = "HKEY_USERS"
  CASE HKEY_PERFORMANCE_DATA: whatKey = "HKEY_PERFORMANCE_DATA"
  CASE HKEY_CURRENT_CONFIG: whatKey = "HKEY_CURRENT_CONFIG"
  CASE HKEY_DYN_DATA: whatKey = "HKEY_DYN_DATA"
END SELECT
END FUNCTION

FUNCTION formatData$ (dwType AS _UNSIGNED LONG, numBytes AS _UNSIGNED LONG, bData AS STRING)
DIM t AS STRING
DIM ul AS _UNSIGNED LONG
DIM b AS _UNSIGNED _BYTE
SELECT CASE dwType
  CASE REG_SZ, REG_EXPAND_SZ, REG_MULTI_SZ
    formatData = LEFT$(bData, numBytes - 1)
  CASE REG_DWORD
    t = LCASE$(HEX$(CVL(LEFT$(bData, 4))))
    formatData = "0x" + STRING$(8 - LEN(t), &H30) + t
  CASE ELSE
    IF numBytes THEN
      b = ASC(LEFT$(bData, 1))
      IF b < &H10 THEN
        t = t + "0" + LCASE$(HEX$(b))
      ELSE
        t = t + LCASE$(HEX$(b))
      END IF
    END IF
    FOR ul = 2 TO numBytes
      b = ASC(MID$(bData, ul, 1))
      IF b < &H10 THEN
        t = t + " 0" + LCASE$(HEX$(b))
      ELSE
        t = t + " " + LCASE$(HEX$(b))
      END IF
    NEXT
    formatData = t
END SELECT
END FUNCTION
```

Code courtesy of Michael Calkins

Note: The names used in a Font Dialog Box and the actual file names are saved to the *F0NTList.INF*(name uses zero) file to be compared with user entries. To check for *Bold* and *Italics* or combined types of font requests see Font Dialog Box Library above.

Note: The above procedure only reads the Registry. Edit or alter the Registry at your own peril!

Windows Registry Libraries

(Return to Table of Contents)

Game Pad

Function *joyGetPosEx* allows for more buttons and dual analog multiple sticks

```
DECLARE DYNAMIC LIBRARY "winmm"
    FUNCTION joyGetNumDevs% () ' Number of joysticks supported on system
    FUNCTION joyGetPosEx& (BYVAL uJoyID AS _UNSIGNED INTEGER, joyref AS LONG)
END DECLARE

TYPE JOYINFOEX
    dwSize AS LONG
    dwFlags AS LONG
    dwXpos AS LONG
    dwYpos AS LONG
    dwZpos AS LONG
    dwRpos AS LONG
    dwUpos AS LONG
    dwVpos AS LONG
    dwButtons AS LONG
    dwButtonNumber AS LONG
    dwPOV AS LONG
    dwReserved1 AS LONG
    dwReserved2 AS LONG
END TYPE

DIM Joy1 AS JOYINFOEX
Joy1.dwSize = LEN(Joy1)

DO
    x& = joyGetPosEx(0, Joy1.dwSize)

    LOCATE 1, 1:
    PRINT Joy1.dwSize
    PRINT Joy1.dwFlags
    PRINT Joy1.dwXpos
    PRINT Joy1.dwYpos
    PRINT Joy1.dwZpos
    PRINT Joy1.dwRpos
    PRINT Joy1.dwUpos
    PRINT Joy1.dwVpos

LOOP UNTIL INKEY$ <> ""
```

Code courtesy of Unseenmachine

```
DECLARE DYNAMIC LIBRARY "winmm"
    FUNCTION joyGetNumDevs% () ' Number of joysticks supported on system
    FUNCTION joyGetPos& (BYVAL uJoyID AS _UNSIGNED INTEGER, joyref AS _UNSIGNED LONG)
END DECLARE

TYPE JOYINFO
    wXpos AS _UNSIGNED LONG
    wYpos AS _UNSIGNED LONG
    wZpos AS _UNSIGNED LONG
    wButtons AS _UNSIGNED LONG
END TYPE

TYPE PadCalibration
    IsAnalog AS INTEGER
    XMin AS LONG
    YMin AS LONG
    XMax AS LONG
    YMax AS LONG
    XNorm AS LONG
    YNorm AS LONG
    Button1 AS INTEGER
    Button2 AS INTEGER
    Button3 AS INTEGER
    Button4 AS INTEGER
    Button5 AS INTEGER
    Button6 AS INTEGER
END TYPE

DIM MyJoyCal AS PadCalibration, MyJoy AS JOYINFO


'// Simple calibration.

PRINT "Press button 1 (A on XBox | X on PS3) "
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons > 0
MyJoyCal.Button1 = MyJoy.wButtons
SLEEP 1

PRINT "Press button 2 (B on XBox | Circle on PS3) "
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons <> 0
MyJoyCal.Button2 = MyJoy.wButtons
SLEEP 1

PRINT "Press button 3 (X on XBox | Square on PS3) "
```

```
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons <> 0
MyJoyCal.Button3 = MyJoy.wButtons
SLEEP 1

PRINT "Press button 4 (Y on XBox | Triangle on PS3) "
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons <> 0
MyJoyCal.Button4 = MyJoy.wButtons
SLEEP 1

PRINT "Press button 5 (R1) "
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons <> 0
MyJoyCal.Button5 = MyJoy.wButtons
SLEEP 1

PRINT "Press button 6 (L1) "
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons <> 0
MyJoyCal.Button6 = MyJoy.wButtons
SLEEP 1

PRINT "Leave the joystick in its central position and press button 1"
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons = MyJoyCal.Button1
MyJoyCal.XNorm = MyJoy.wXpos
SLEEP 1

PRINT "Push the joystick as far left as possible and press button 1"
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons = MyJoyCal.Button1
MyJoyCal.XMin = MyJoy.wXpos
SLEEP 1

PRINT "Push the joystick as far right as possible and press button 1"
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons = MyJoyCal.Button1
MyJoyCal.XMax = MyJoy.wXpos
SLEEP 1

PRINT "Push the joystick as far up as possible and press button 1"
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons = MyJoyCal.Button1
MyJoyCal.YMin = MyJoy.wXpos
SLEEP 1

PRINT "Push the joystick as far down as possible and press button 1"
DO
    d% = JoyButtons(0, MyJoy, MyJoyCal)
LOOP UNTIL MyJoy.wButtons = MyJoyCal.Button1
MyJoyCal.YMax = MyJoy.wXpos
SLEEP 1

CLS

'//How to get the gamepads status.

DO
    a% = StickXPos(0, MyJoy, MyJoyCal)
    b% = StickYPos(0, MyJoy, MyJoyCal)
    c% = StickZPos(0, MyJoy, MyJoyCal)
    d% = JoyButton1(0, MyJoy, MyJoyCal)
    e% = JoyButton2(0, MyJoy, MyJoyCal)
    f% = JoyButton3(0, MyJoy, MyJoyCal)
    g% = JoyButton4(0, MyJoy, MyJoyCal)
    h% = JoyButton5(0, MyJoy, MyJoyCal)
    i% = JoyButton6(0, MyJoy, MyJoyCal)

    LOCATE 1, 1: PRINT "X axis : ", MyJoy.wXpos
    PRINT "Y axis : ", MyJoy.wYpos
    PRINT "Z axis : ", MyJoy.wZpos
    PRINT "Button 1 : ", d%
    PRINT "Button 2 : ", e%
    PRINT "Button 3 : ", f%
    PRINT "Button 4 : ", g%
    PRINT "Button 5 : ", h%
    PRINT "Button 6 : ", i%

LOOP UNTIL INKEY$ <> ""


'// Functions

FUNCTION StickXPos (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
StickXPos = Joyref.wXpos
END FUNCTION

FUNCTION StickYPos (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
StickYPos = Joyref.wYpos
END FUNCTION
```

```
FUNCTION StickZPos (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
StickZPos = Joyref.wZpos
END FUNCTION

FUNCTION JoyButtons (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
JoyButtons = Joyref.wButtons
END FUNCTION

FUNCTION JoyButton1 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button1 THEN
    JoyButton1 = -1
ELSE
    JoyButton1 = 0
END IF
END FUNCTION

FUNCTION JoyButton2 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button2 THEN
    JoyButton2 = -1
ELSE
    JoyButton2 = 0
END IF
END FUNCTION

FUNCTION JoyButton3 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button3 THEN
    JoyButton3 = -1
ELSE
    JoyButton3 = 0
END IF
END FUNCTION

FUNCTION JoyButton4 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button4 THEN
    JoyButton4 = -1
ELSE
    JoyButton4 = 0
END IF
END FUNCTION

FUNCTION JoyButton5 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button5 THEN
    JoyButton5 = -1
ELSE
    JoyButton5 = 0
END IF
END FUNCTION

FUNCTION JoyButton6 (Index AS _UNSIGNED INTEGER, Joyref AS JOYINFO, JoyCal AS PadCalibration)
x = joyGetPos(Index, Joyref.wXpos)
IF Joyref.wButtons = JoyCal.Button6 THEN
    JoyButton6 = -1
ELSE
    JoyButton6 = 0
END IF
END FUNCTION
```

(Return to Table of Contents)

Hot Keys (maximize)

Maximizing a minimized program window not in focus using Shift + A as read by the Windows *GetKeyState* function.

```
'=================
'CHEAPO-HOTKEY.BAS
'=================
'Uses GetKeyState Win API to monitor a Key state.
'This demo will maximize the window when Shift+A is pressed at any time.

DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION FindWindowA& (BYVAL ClassName AS _OFFSET, WindowName$) 'handle by title
  FUNCTION GetKeyState% (BYVAL nVirtKey AS LONG) 'reads Windows key presses independently
  FUNCTION ShowWindow& (BYVAL hwnd AS LONG, BYVAL nCmdShow AS LONG) 'minimize or maximize
END DECLARE

title$ = "Cheapo Hotkey (Shift+A)" 'string variable avoids title typo's
_TITLE title$
hwnd& = FindWindowA(0, title$ + CHR$(0))

PRINT "Minimize this window, then Press Shift+A to bring it back up."

'=== below minimizes it for you
'_DELAY 4
'x& = ShowWindow&(hwnd&, 2)
'============================

DO
```

```
  IF GetKeyState(16) < 0 AND GetKeyState(ASC("A")) < 0 THEN '<==== Shift+A
      y& = ShowWindow&(hwnd&, 1)
      PRINT "That is all. Hoped it worked."; x&; y&
      END
  END IF

  _LIMIT 30 'Don't be a hog
LOOP
```

Code courtesy of Dav

Note: Virtual Hot keys can be used when a QB64 program is **not in focus** too! See **Focus** to bring a QB64 program into focus.

```
'                                Virtual KeyState Codes
'
' Esc  F1  F2  F3  F4  F5  F6  F7  F8  F9  F10  F11 F12   Sys ScL Pause
' 27   112 113 114 115 116 117 118 119 120 121  122 123   44  145 19
' `~  1!  2@  3#  4$  5%  6^  7&  8* 9(  0) -_  =+  BkS   Ins Hme PUp   NumL  /   *    -
' 192 49  50  51  52  53  54  55  56  57 48 189 187  8     45  36  33    144  111  106 109
' Tab Q   W   E   R   T   Y   U   I   O   P  [{  ]}  \|   Del End PDn   7Hme 8/▲  9PU  +
'  9  81  87  69  82  84  89  85  73  79  80 219 221 120   46  35  34    103  104  105 107
' CapL  A   S   D   F   G   H   J   K   L   ;:  '" Enter                 4/◄-  5   6/-► E
'  20   65  83  68  70  71  72  74  75  76 186 222  13                   100  101  102  n
' Shift  Z   X   C   V   B   N   M   ,<  .>  /?   Shift       ▲         1End 2/▼  3PD  t
' 16/160 90  88  67  86  66  78  77 188  190 191   16/161      38        97   98   99   e
' Ctrl   Win   Alt       Spacebar      Alt Win Menu Ctrl   ◄-  ▼   -►   0Ins      .Del  r
' 17/162 91  18/164        32        18/165 92  93 17/163  37  40  39   96         110  13
'
'     Num Lock On values shown. Off values same as functions and arrows, 5 = code 12.
'
'  Mouse click returns: LB = 1, RB = 2, MB = 4, etc. Special keys (http://msdn.microsoft.com/en-us/library/win
```

NOTE: The above commented table can be copied and pasted directly into the QB64 IDE

Invisible key and mouse logger that does not require program focus. Press ESCape key to view the logged activity.

```
$CONSOLE
_DEST _CONSOLE ' for demonstration only
PRINT "The actual program is hiding. Type some stuff!"
PRINT
PRINT "Press ESC to quit logging keys and see results.":
_DEST 0
'====================== End Demo Console Code ==============================

_SCREENHIDE 'makes the program invisible to the user. Escape key displays log!

DECLARE LIBRARY  'function is already used by QB64 so "User32" is not required
  FUNCTION GetAsyncKeyState% (BYVAL vkey AS LONG)
END DECLARE

DIM theitem$(1000)

DO: _LIMIT 100
  FOR thekey = &H30 TO &H5A
    IF GetAsyncKeyState(thekey) THEN
      theitem$(a) = theitem$(a) + CHR$(thekey)
      DO
      LOOP UNTIL GetAsyncKeyState(thekey) = 0
    END IF
  NEXT
  IF GetAsyncKeyState(1) THEN
    a = a + 1
    theitem$(a) = "{MOUSE LEFT}"
    DO
    LOOP UNTIL GetAsyncKeyState(1) = 0
  END IF
  IF GetAsyncKeyState(2) THEN
    a = a + 1
    theitem$(a) = "{MOUSE RIGHT}"
    DO
    LOOP UNTIL GetAsyncKeyState(2) = 0
  END IF
  IF GetAsyncKeyState(4) THEN
    a = a + 1
    theitem$(a) = "{MOUSE MIDDLE}"
    DO
    LOOP UNTIL GetAsyncKeyState(4) = 0
  END IF
  IF GetAsyncKeyState(8) THEN
    theitem$(a) = theitem$(a) + "{BS}"
    DO
    LOOP UNTIL GetAsyncKeyState(8) = 0
  END IF
  IF GetAsyncKeyState(9) THEN
    theitem$(a) = theitem$(a) + "{TAB}"
    DO
    LOOP UNTIL GetAsyncKeyState(9) = 0
  END IF
  IF GetAsyncKeyState(&HD) THEN
    a = a + 1
    theitem$(a) = "{ENTER}"
    DO
    LOOP UNTIL GetAsyncKeyState(&HD) = 0
  END IF
```

```
  IF GetAsyncKeyState(&H14) THEN
    theitem$(a) = theitem$(a) + "{CAPS LOCK}"
    DO
    LOOP UNTIL GetAsyncKeyState(&H14) = 0
  END IF
  IF GetAsyncKeyState(&H20) THEN
    theitem$(a) = theitem$(a) + " "
    DO
    LOOP UNTIL GetAsyncKeyState(&H20) = 0
  END IF
  IF GetAsyncKeyState(&H1B) THEN
    theitem$(a) = theitem$(a) + "{ESC}"
    EXIT DO 'ESC key exits loop and prints logged key presses
  END IF
LOOP

_SCREENSHOW 'makes program visible with ESC key press
FOR b = 0 TO a
  PRINT theitem$(b)
  IF b MOD 20 = 19 THEN COLOR 12: PRINT "press any key": SLEEP: COLOR 7
NEXT
```

Code by Cyperium

Note: The program will run invisibly without a program icon appearing in the task bar until the ESC key is pressed and the log will be displayed.

(Return to Table of Contents)

Keyboard Lock Settings

Change Cap Lock, Scroll Lock and Number Lock settings and respective lights or onscreen indicators.

```
'public domain, 2012 april, michael calkins
CONST INPUT_KEYBOARD = 1
CONST KEYEVENTF_KEYDOWN = 0
CONST KEYEVENTF_KEYUP = &H2

CONST VK_CAPITAL = &H14
CONST VK_NUMLOCK = &H90
CONST VK_SCROLL = &H91

CONST scCapital = &H3A
CONST scNumlock = &H45
CONST scScroll = &H46

DECLARE DYNAMIC LIBRARY "kernel32"
 FUNCTION GetLastError~& ()
END DECLARE

DECLARE DYNAMIC LIBRARY "user32"
 FUNCTION SendInput~& (BYVAL nInputs~&, BYVAL pInputs%&, BYVAL cbSize&)
 FUNCTION GetMessageExtraInfo%& ()
END DECLARE

TYPE KeyInputStruc
 type AS _UNSIGNED LONG
 wVk AS _UNSIGNED INTEGER
 wScan AS _UNSIGNED INTEGER
 dwFlags AS _UNSIGNED LONG
 time AS _UNSIGNED LONG
 dwExtraInfo AS _OFFSET
 pad0 AS LONG 'to accomodate the size of a MOUSEINPUT
 pad1 AS LONG
END TYPE

DIM ki(0 TO 1) AS KeyInputStruc '2 index array will be zero initialized

ki(0).type = INPUT_KEYBOARD
ki(1).type = INPUT_KEYBOARD
ki(0).dwFlags = KEYEVENTF_KEYDOWN
ki(1).dwFlags = KEYEVENTF_KEYUP
ki(0).dwExtraInfo = GetMessageExtraInfo 'read function value
ki(1).dwExtraInfo = ki(0).dwExtraInfo   'assign same value

RANDOMIZE TIMER
DO UNTIL INKEY$ = CHR$(&H1B)
  _DELAY 2
 SELECT CASE INT(RND * 3)
  CASE 0
   ki(0).wVk = VK_NUMLOCK
   ki(1).wVk = VK_NUMLOCK
   ki(0).wScan = scNumlock
   ki(1).wScan = scNumlock
   PRINT "Num lock..."
  CASE 1
   ki(0).wVk = VK_CAPITAL
   ki(1).wVk = VK_CAPITAL
   ki(0).wScan = scCapital
   ki(1).wScan = scCapital
   PRINT "Caps lock..."
  CASE 2
   ki(0).wVk = VK_SCROLL
   ki(1).wVk = VK_SCROLL
   ki(0).wScan = scScroll
   ki(1).wScan = scScroll
   PRINT "Scroll lock..."
```

```
 END SELECT
 l = SendInput(2, _OFFSET(ki(0)), LEN(ki(0))) '2 tells function to read two indices(0 and 1)
 IF l <> 2 THEN
  PRINT l
  PRINT "0x" + LCASE$(HEX$(GetLastError))
 END IF
LOOP
END
```

Code by Michael Calkins

Note: In QB64 the number pad lock does not affect the INP(&H60) release code returns of the extended keys like it did in Qbasic.

(Return to Table of Contents)

Message Box

```
'Message Box Constant values as defined by Microsoft (MBType)
CONST MB_OK& = 0                 'OK button only
CONST MB_OKCANCEL& = 1           'OK & Cancel
CONST MB_ABORTRETRYIGNORE& = 2   'Abort, Retry & Ignore
CONST MB_YESNOCANCEL& = 3        'Yes, No & Cancel
CONST MB_YESNO& = 4              'Yes & No
CONST MB_RETRYCANCEL& = 5        'Retry & Cancel
CONST MB_CANCELTRYCONTINUE& = 6 'Cancel, Try Again & Continue
CONST MB_ICONSTOP& = 16          'Error stop sign icon
CONST MB_ICONQUESTION& = 32      'Question-mark icon
CONST MB_ICONEXCLAMATION& = 48   'Exclamation-point icon
CONST MB_ICONINFORMATION& = 64   'Letter i in a circle icon
CONST MB_DEFBUTTON1& = 0         '1st button default(left)
CONST MB_DEFBUTTON2& = 256       '2nd button default
CONST MB_DEFBUTTON3& = 512       '3rd button default(right)
CONST MB_APPLMODAL& = 0          'Message box applies to application only
CONST MB_SYSTEMMODAL& = 4096     'Message box on top of all other windows
CONST MB_SETFOCUS& = 65536       'Set message box as focus
CONST IDOK& = 1                  'OK button pressed
CONST IDCANCEL& = 2              'Cancel button pressed
CONST IDABORT& = 3               'Abort button pressed
CONST IDRETRY& = 4               'Retry button pressed
CONST IDIGNORE& = 5              'Ignore button pressed
CONST IDYES& = 6                 'Yes button pressed
CONST IDNO& = 7                  'No button pressed
CONST IDTRYAGAIN& = 10           'Try again button pressed
CONST IDCONTINUE& = 1            'Continue button pressed
'-------------------------------------------------------------------------------------------

DECLARE LIBRARY
FUNCTION MessageBox& (BYVAL Zer0 AS LONG, Message AS STRING, Title AS STRING, BYVAL MBType AS _UNSIGNED LONG)
END DECLARE

DO
  msg& = 0: icon& = 0: DB& = 0
  INPUT "Enter Message Box type(0 to 6 other Quits): ", BOX&
  IF BOX& < 0 OR BOX& > 6 THEN EXIT DO

  INPUT "Enter Icon&(0=none, 1=stop, 2=?, 3=!, 4=info): ", Icon&

  IF BOX& THEN INPUT "Enter Default Button(1st, 2nd or 3rd): ", DB&
  IF DB& THEN DB& = DB& - 1     'adjust value to 0, 1, or 2
  msg& = MsgBox&("Box Title", "Box text message", BOX&, Icon&, DB&, 4096) 'on top of all windows

  PRINT "Button ="; msg&
LOOP
END

FUNCTION MsgBox& (Title$, Message$, BoxType&, Icon&, DBtn&, Mode&)
SELECT CASE Icon&
  CASE 1: Icon& = MB_ICONSTOP&          'warning X-sign icon
  CASE 2: Icon& = MB_ICONQUESTION&      'question-mark icon
  CASE 3: Icon& = MB_ICONEXCLAMATION&   'exclamation-point icon
  CASE 4: Icon& = MB_ICONINFORMATION&   'lowercase letter i in circle
  CASE ELSE: Icon& = 0 'no icon
END SELECT
IF BoxType& > 0 AND DBtn& > 0 THEN 'set default button as 2nd(256) or 3rd(512)
  SELECT CASE BoxType&
    CASE 2, 3, 6
     IF DBtn& = 2 THEN Icon& = Icon& + MB_DEFBUTTON3& ELSE Icon& = Icon& + MB_DEFBUTTON2& '3 button
    CASE ELSE: Icon& = Icon& + MB_DEFBUTTON2& '2nd button default
  END SELECT
END IF
Focus& = MB_SetFocus&
MsgBox& = MessageBox&(0, Message$, Title$, BoxType& + Icon& + Mode& + Focus&) 'focus on button
END FUNCTION
```

Note: The demo above can show all of the possible message box options. The actual code necessary is quite simple.

(Return to Table of Contents)

Mouse Area

Program limits the mouse to a box portion of the Windows desktop using the Rectangle TYPE to define the mouse area.

```
TYPE Rectangle
  left AS LONG
  top AS LONG
  right AS LONG
  bottom AS LONG
END TYPE
DIM Rec AS Rectangle


DECLARE DYNAMIC LIBRARY "User32"
  FUNCTION ClipCursor%% (Rect AS Rectangle) 'sets mouse box work area on desktop
  SUB SetCursorPos (BYVAL x AS LONG, BYVAL y AS LONG) 'move cursor position
END DECLARE

SCREEN _NEWIMAGE(320, 200, 32)
SetCursorPos 40, 36  'move cursor to left side of desktop

PRINT "Press a key and the mouse is boxed in!"
K$ = INPUT$(1)

Rec.left = 600
Rec.top = 400
Rec.bottom = 700
Rec.right = 800
work%% = ClipCursor(Rec)

CLS
PRINT work%%
PRINT "Click the mouse and window to quit!"
DO
  m = _MOUSEINPUT

LOOP UNTIL _MOUSEBUTTON(2) OR _MOUSEBUTTON(1)
SetCursorPos 40, 36 'attempts to move mouse to left

SYSTEM
```

Note: The left and top positions must be less than the bottom and right pixel position values. Click mouse to exit box area. Click program window to quit.

(Return to Table of Contents)

Open another Program

```
'Uses Kernel32 WinAPI to execute a program in a QB64 program.  Coded by Dav

DECLARE LIBRARY
  Function WinExec (lpCmdLine AS STRING, BYVAL nCmdShow AS LONG)
END DECLARE

Winmode% = 1

'0 = Hides the window and activates another window.
'1 = Activates and displays a normal sized window.
'2 = Activates the window and minimized to taskbar.
'3 = Activates the window and displays it as a maximized window.

'NOTE: If you do 0 (hide), you'll have to Kill the process using your TaskManager!!!

'=== Open notepad and load samples.txt in the QB64 directory

Filename$ = "notepad.exe samples.txt" + CHR$(0)

'NOTE: EXE filename must be a NULL terminates..CHR$(0)...

Result = WinExec(Filename$, Winmode%)

'=== Show results ...

SELECT CASE Result
   CASE 0: PRINT "System out of memory or resources."
   CASE 2: PRINT "The specified file was not found."
   CASE 3: PRINT "The specified path was not found."
   CASE 11: PRINT "The file is invalid (non-Win32 .EXE or error in .EXE image)."
   CASE IS > 31: PRINT "Program opened normally."
   CASE ELSE: PRINT "Unknown error: "; Result
END SELECT
END
```

Code by Dav

Note: The Library file can only run valid Windows executable programs. Not DOS console EXE programs!

(Return to Table of Contents)

Play WAV Sounds

```
CONST SND_SYNC = 0 'Windows controlled
CONST SND_ASYNC = 1 'user controlled
CONST SND_NODEFAULT = 2 'only plays sound file requested
CONST SND_LOOP = 8 'loops the sound. Use ASYNC also to stop later
CONST SND_NOSTOP = &H10 'does not allow a sound to be stopped
CONST SND_NOWAIT = &H2000 'will not play sound if driver is busy
CONST SND_PURGE = &H40 'stop any sound playing

DECLARE DYNAMIC LIBRARY "winmm"
  FUNCTION PlaySound% ALIAS PlaySoundA (lpszName AS STRING, BYVAL hModule AS INTEGER, BYVAL dwFlags AS INTEGER)
END DECLARE

LINE INPUT "Enter WAV sound file name: ", FileName$
PRINT "Play asynchronously?(Y/N) ";
K$ = UCASE$(INPUT$(1))
PRINT K$
IF K$ = "Y" THEN Synch = SND_ASYNC ELSE Synch = SND_SYNC

retval% = PlaySound(FileName$, 0, Synch)
```

Code by Ted Weissgerber

Note: ASYNC allows the program to stop the sound by sending a null file name. Flag constants can be added so loop and ASYNC would total 9.

WINMM.DLL Functions (http://msdn.microsoft.com/en-us/library/ms712636)

(Return to Table of Contents)

Run One Instance

Program catches another instance of the program descriptor label running and closes it

```
CONST ERROR_ALREADY_EXISTS = &HB7

DECLARE DYNAMIC LIBRARY "kernel32"
 FUNCTION CreateMutexA%& (BYVAL lpMutexAttributes%&, BYVAL bInitialOwner&, BYVAL lpName%&)
 FUNCTION GetLastError~& ()
END DECLARE

DIM t AS STRING

t = "Global\Some name unique to your program" + CHR$(0)
' see: http://msdn.microsoft.com/en-us/library/ms682411%28v=vs.85%29

IF 0 = CreateMutexA(0, 0, _OFFSET(t)) THEN
 PRINT "CreateMutexA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
 END
END IF
IF ERROR_ALREADY_EXISTS = GetLastError THEN
 PRINT "Sorry. There can be only one."
 SLEEP 2
 SYSTEM
END IF

PRINT "But the most wonderful thing about tiggers is I'm the only one."
DO UNTIL INKEY$ = CHR$(&H1B)
LOOP
SYSTEM
```

Code courtesy of Michael Calkins

Explanation: The same identification string is used in both instances of the running program. Compile example, run the EXE and try to compile and run another instance. 'Sorry. There can be only one.' will be printed to the screen and then it will close.

(Return to Table of Contents)

Send Keys

```
'============
'SENDKEYS.BAS
'============
'A kind of SENDKEYS clone for QB64. Sends keystrokes to active application.
'Coded for QB64 by Dav, JAN/2013


DECLARE DYNAMIC LIBRARY "user32"
    SUB SENDKEYS ALIAS keybd_event (BYVAL bVk AS LONG, BYVAL bScan AS LONG, BYVAL dwFlags AS LONG, BYVAL dwExt
END DECLARE
```

```
CONST KEYEVENTF_KEYUP = &H2
CONST VK_SNAPSHOT = &H2C 'PrtScn key
CONST VK_MENU = &H12 'Alt key
CONST VK_SHIFT = &H10 'Shift key
CONST VK_LWIN = &H5B

PRINT ""
PRINT "SENDKEYS clone example.  Press ENTER to begin..."
PRINT

WHILE INKEY$ <> CHR$(13): WEND

'=== Capture Active window to Clipoard as image (Like Alt+PrtSc)
SENDKEYS VK_MENU, 0, 0, 0
SENDKEYS VK_SNAPSHOT, 0, 0, 0
SENDKEYS VK_SNAPSHOT, 0, KEYEVENTF_KEYUP, 0
SENDKEYS VK_MENU, 0, KEYEVENTF_KEYUP, 0

PRINT "SENDKEYS just sent a Alt+PrtSc keyboard press."
PRINT "A screen-cap of this EXE WINDOW was just sent to your clipboard."
PRINT "Go ahead, check it now, then come back here and press ENTER..."

WHILE INKEY$ <> CHR$(13): WEND

'=== Capture Desktop screen to Clipoard as image (Like Shift+PrtSc)
SENDKEYS VK_SHIFT, 0, 0, 0
SENDKEYS VK_SNAPSHOT, 0, 0, 0
SENDKEYS VK_SNAPSHOT, 0, KEYEVENTF_KEYUP, 0
SENDKEYS VK_SHIFT, 0, KEYEVENTF_KEYUP, 0

PRINT
PRINT "Now SENDKEYS just sent a Shift+PrtSc keyboard press."
PRINT "A screen-cap of your DESKTOP was just sent to your clipboard."
PRINT "Go ahead, check it now, then come back here and press ENTER..."
PRINT

WHILE INKEY$ <> CHR$(13): WEND

PRINT "Now 'Notepad.exe' will open and SENDKEYS will send text to it."
PRINT "Press ENTER to do that now, then close notepad, come back here,"
PRINT "and press ENTER to continue..."

WHILE INKEY$ <> CHR$(13): WEND

SHELL _DONTWAIT "notepad.exe"

SLEEP 1 'give time to make notepad the active window...

'=== Say Hi...
SENDKEYS &H48, 0, 0, 0: SENDKEYS &H48, 0, KEYEVENTF_KEYUP, 0 'H
SENDKEYS &H49, 0, 0, 0: SENDKEYS &H49, 0, KEYEVENTF_KEYUP, 0 'i

PRINT
PRINT "Now press ENTER for SENDKEYS to Minimize all your windows..."
PRINT

WHILE INKEY$ <> CHR$(13): WEND

'=== Minimize windows shortcut
SENDKEYS VK_LWIN, 0, 0, 0
SENDKEYS &H4D, 0, 0, 0
SENDKEYS VK_LWIN, 0, KEYEVENTF_KEYUP, 0

PRINT "That's all. have a nice day."

END
```

Code courtesy of Dav

```
'                                Virtual KeyState Codes
'
' Esc  F1  F2  F3  F4  F5  F6  F7  F8  F9  F10  F11 F12   Sys ScL Pause
' 27   112 113 114 115 116 117 118 119 120 121  122 123   44  145 19
' `~  1!  2@  3#  4$  5%  6^  7&  8* 9(  0) -_  =+  BkS   Ins Hme PUp   NumL  /   *    -
' 192 49  50  51  52  53  54  55  56  57 48 189 187  8    45  36  33    144  111  106 109
' Tab Q   W   E   R   T   Y   U   I   O   P  [{  ]}  \|   Del End PDn   7Hme 8/▲ 9PU  +
'  9  81  87  69  82  84  89  85  73  79  80 219 221 120  46  35  34    103  104  105 107
' CapL  A   S   D   F   G   H   J   K   L   ;:  '" Enter                4/◄- 5   6/-► E
'  20   65  83  68  70  71  72  74  75  76 186 222  13                   100  101  102  n
' Shift  Z   X   C   V   B   N   M   ,<  .>  /?   Shift       ▲        1End 2/▼ 3PD  t
' 16/160 90  88  67  86  66  78  77 188  190 191   16/161      38       97   98   99   e
' Ctrl   Win   Alt       Spacebar      Alt Win Menu Ctrl   ◄-  ▼   -►   0Ins     .Del  r
' 17/162 91  18/164        32        18/165 92  93 17/163  37  40  39   96        110  13
'
'     Num Lock On values shown. Off values same as functions and arrows, 5 = code 12.
'
'  Mouse click returns: LB = 1, RB = 2, MB = 4, etc. Special keys (http://msdn.microsoft.com/en-us/library/win
```

NOTE: The above commented table can be copied and pasted directly into the QB64 IDE

(Return to Table of Contents)

System Metrics

Returns the dimensions of the current user's desktop. Can be used to get various Windows settings.

```
CONST SM_CXSCREEN = 0 'Width of user desktop
CONST SM_CYSCREEN = 1 'Height of user desktop
CONST SM_CXFULLSCREEN = 16 ' Width of window client area
CONST SM_CYFULLSCREEN = 17 ' Height of window client area
CONST SM_CYMENU = 15 ' Height of menu
CONST SM_CYCAPTION = 4 ' Height of caption or title
CONST SM_CXFRAME = 32 ' Width of window frame
CONST SM_CYFRAME = 33 ' Height of window frame
CONST SM_CXHSCROLL = 21 ' Width of arrow bitmap on horizontal scroll bar
CONST SM_CYHSCROLL = 3 ' Height of arrow bitmap on horizontal scroll bar
CONST SM_CXVSCROLL = 2 ' Width of arrow bitmap on vertical scroll bar
CONST SM_CYVSCROLL = 20 ' Height of arrow bitmap on vertical scroll bar
CONST SM_CXSIZE = 30 ' Width of bitmaps in title bar
CONST SM_CYSIZE = 31 ' Height of bitmaps in title bar
CONST SM_CXCURSOR = 13 ' Width of cursor
CONST SM_CYCURSOR = 14 ' Height of cursor
CONST SM_CXBORDER = 5 ' Width of window frame that cannot be sized
CONST SM_CYBORDER = 6 ' Height of window frame that cannot be sized
CONST SM_CXDOUBLECLICK = 36 ' Width of rectangle around the location of the first click.
CONST SM_CYDOUBLECLICK = 37 ' Height of rectangle around the location of the first click.
CONST SM_CXDLGFRAME = 7 ' Width of dialog frame window
CONST SM_CYDLGFRAME = 8 ' Height of dialog frame window
CONST SM_CXICON = 11 ' Width of icon
CONST SM_CYICON = 12 ' Height of icon
CONST SM_CXICONSPACING = 38 ' Width of rectangles the system uses to position tiled icons
CONST SM_CYICONSPACING = 39 ' Height of rectangles the system uses to position tiled icons
CONST SM_CXMIN = 28 ' Minimum width of window
CONST SM_CYMIN = 29 ' Minimum height of window
CONST SM_CXMINTRACK = 34 ' Minimum tracking width of window
CONST SM_CYMINTRACK = 35 ' Minimum tracking height of window
CONST SM_CXHTHUMB = 10 ' Width of scroll box (thumb) on horizontal scroll bar
CONST SM_CYVTHUMB = 9 ' Width of scroll box (thumb) on vertical scroll bar
CONST SM_DBCSENABLED = 42 ' Returns a non-zero if the current Windows version uses double-byte characters, othe
CONST SM_DEBUG = 22 ' Returns non-zero if the Windows version is a debugging version
CONST SM_MENUDROPALIGNMENT = 40 'Alignment of pop-up menus.
CONST SM_MOUSEPRESENT = 19 ' Non-zero if mouse hardware is installed
CONST SM_PENWINDOWS = 41 ' Handle of Pen Windows dynamic link library if Pen Windows is installed
CONST SM_SWAPBUTTON = 23 ' Non-zero if the left and right mouse buttons are swapped

DECLARE LIBRARY
    FUNCTION GetSystemMetrics& (BYVAL n AS LONG)
END DECLARE

PRINT trimstr$(GetSystemMetrics(SM_CXSCREEN)); "x"; trimstr$(GetSystemMetrics(SM_CYSCREEN))

s& = _SCREENIMAGE
PRINT _WIDTH(s&); "X"; _HEIGHT(s&)

END 3

FUNCTION trimstr$ (whatever)
trimstr = LTRIM$(RTRIM$(STR$(whatever)))
END FUNCTION
```

(Return to Table of Contents)

Top Most Window

This Windows only procedure will make the program window always stay on top of other windows, but not always in focus. (See Windows Focus)

```
'public domain

CONST SWP_NOSIZE = &H0001      'ignores cx and cy size parameters
CONST SWP_NOMOVE = &H0002      'ignores x and y position parameters
CONST SWP_NOZORDER = &H0004    'keeps z order and ignores hWndInsertAfter parameter
CONST SWP_NOREDRAW = &H0008    'does not redraw window changes
CONST SWP_NOACTIVATE = &H0010  'does not activate window
CONST SWP_FRAMECHANGED = &H0020
CONST SWP_SHOWWINDOW = &H0040
CONST SWP_HIDEWINDOW = &H0080
CONST SWP_NOCOPYBITS = &H0100
CONST SWP_NOOWNERZORDER = &H0200
CONST SWP_NOSENDCHANGING = &H0400
CONST SWP_DRAWFRAME = SWP_FRAMECHANGED
CONST SWP_NOREPOSITION = SWP_NOOWNERZORDER
CONST SWP_DEFERERASE = &H2000
CONST SWP_ASYNCWINDOWPOS = &H4000
CONST HWND_TOP = 0        'window at top of z order no focus
CONST HWND_BOTTOM = 1     'window at bottom of z order no focus
CONST HWND_TOPMOST = -1   'window above all others no focus unless active
CONST HWND_NOTOPMOST = -2 'window below active no focus

DECLARE DYNAMIC LIBRARY "user32"
 FUNCTION FindWindowA%& (BYVAL lpClassName%&, BYVAL lpWindowName%&)
 FUNCTION SetWindowPos& (BYVAL hWnd%&, BYVAL hWndInsertAfter%&, BYVAL X&, BYVAL Y&, BYVAL cx&, BYVAL cy&, BYVAI
 FUNCTION GetForegroundWindow%&
END DECLARE
```

```
DECLARE DYNAMIC LIBRARY "kernel32"
 FUNCTION GetLastError~& ()
END DECLARE

DIM t AS STRING
DIM hWnd AS _OFFSET

RANDOMIZE TIMER
t = HEX$(RND * &H1000000) 'random title for FindWindowA
_TITLE t
t = t + CHR$(0)
hWnd = FindWindowA(0, _OFFSET(t))
_TITLE "This Window will always be on Top" 'any title

_DELAY 4 'delay allows user to click focus on other windows

'set as topmost window and move without sizing or activation
IF 0 = SetWindowPos(hWnd, HWND_TOPMOST, 200, 200, 0, 0, SWP_NOSIZE OR SWP_NOACTIVATE) THEN
 PRINT "SetWindowPos failed. 0x" + LCASE$(HEX$(GetLastError))
END IF

x%& = GetForegroundWindow%& 'find currently focused process handle

PRINT "Program handle:"; hWnd; "Focus handle:"; x%&
IF hWnd <> x%& THEN _SCREENCLICK 240, 240 'add 40 to x and y to focus on positioned window

END
```

Adapted from code by Michael Calkins

Explanation: When other windows are clicked on, this program window will stay on top. Click it to return focus.

SetWindowPos can also move the window's TLC corner position and re-size the window when not flagged.
When the window is moved to a position, _SCREENCLICK can be used to focus on the program window.

***SetForegroundWindow* will not work with the *SetWindowPos*!**

http://msdn.microsoft.com/en-us/library/ms633545%28v=vs.85%29

(Return to Table of Contents)

Video File Player

Video player for MPG or AVI video files:

```
DECLARE DYNAMIC LIBRARY "WINMM"
    FUNCTION mciSendStringA% (lpstrCommand AS STRING, lpstrReturnString AS STRING, BYVAL uReturnLength AS INTEG
    ' mciSendStringA function plays media files and returns the following:
    ' 0 = command sucessful
    ' - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
    ' lpstrCommand is the MCI command string (and optional flags) to send.
    ' lpstrReturnString is a string that holds any return information.
    ' uReturnLength is the length of the lpstrReturnString string passed.
    ' NOTE: If lpstrCommand given doesn't retun a value then lpstrReturnString
    '       can be empty and uReturnLength can be set to 0.
    ' hwndCallback contains a callback window handle (only if the Notify flag used in lpstrCommand)
    '====================================================================
    FUNCTION mciGetErrorStringA% (BYVAL dwError AS INTEGER, lpstrBuffer AS STRING, BYVAL uLength AS INTEGER)
    ' mciGetErrorStringA returns error info if the mciSendStringA failed.
    ' dwError is the return value from the mciSendString function.
    ' lpstrBuffer string holds the error information returned by the function.
    ' uLength is the length of the lpstrBuffer string buffer.
    '====================================================================
END DECLARE

DECLARE CUSTOMTYPE LIBRARY
    FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$)
END DECLARE


handle& = _NEWIMAGE(800, 600, 256)
SCREEN handle&


_TITLE "QB64 Video"
hwnd& = FindWindow(0, "QB64 Video" + CHR$(0))

ReturnString$ = SPACE$(255)
ErrorString$ = SPACE$(255)
filename$ = "c:\DavPiano.mpg" '<========== video file to play

a% = mciSendStringA%("open " + filename$ + " style popup", ReturnString$, LEN(ReturnString$), 0)

IF a% THEN
    x% = mciGetErrorStringA%(a%, ErrorString$, LEN(ErrorString$))
    PRINT ErrorString$
    END
ELSE
    a2% = mciSendStringA%("window " + filename$ + " handle " + STR$(hwnd&), ReturnString$, LEN(ReturnString$),
    b% = mciSendStringA%("play " + filename$, "", 0, 0)
    _SCREENMOVE _MIDDLE
    '=== Play video...
    DO: _LIMIT 30: LOOP UNTIL INKEY$ <> ""
```

```
    x% = mciSendStringA%("stop " + filename$, "", 0, 0)
    x% = mciSendStringA%("close " + filename$, "", 0, 0)
END IF
```

Code courtesy of Dav

Web Page Download

Downloads the contents of a web page as an HTML or text file. Text can be edited for page information.

```
DECLARE DYNAMIC LIBRARY "urlmon"
  FUNCTION URLDownloadToFileA% (BYVAL pCaller AS LONG, szURL AS STRING, szFileName AS STRING, BYVAL dwReserved
END DECLARE

'=== URL to grab (page or a file)
URL$ = "http://www.qbasicnews.com/dav/"
'=== File to save URL as
URLfile$ = "DavsIndex.html"

'=== Download it.  Returns 0 if succeeded
a% = URLDownloadToFileA%(0, URL$, URLfile$, 0, 0)

PRINT "Grabbing : "; URL$: PRINT "Saving as: "; URLfile$
PRINT "Sleeping 7 secs to do the deed..."
SLEEP 7
PRINT a%
```

Code courtesy of Dav

(Return to Table of Contents)

Windows API

Program finds the window handle by the title and then does various things with the window using that handle _OFFSET value.

```
CONST HWND_BOTTOM = 1     'places the window at the bottom of the Z order
CONST HWND_TOP = 0        'places the window at the top of the Z order
CONST HWND_TOPMOST = -1   'places the window above all non-topmost windows
CONST HWND_NOTOPMOST = -2 'places the window behind all topmost windows

DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION FindWindowA%& (BYVAL class AS _OFFSET, Title$)
  FUNCTION CloseWindow& (BYVAL hwnd AS _OFFSET)
  FUNCTION OpenIcon& (BYVAL hwnd AS _OFFSET)
  FUNCTION SetWindowTextA& (BYVAL hwnd AS _OFFSET, NewTitle$)
  FUNCTION GetWindowThreadProcessId& (BYVAL hwnd AS _OFFSET, BYVAL null AS LONG)
  FUNCTION SetWindowPos& (BYVAL hwnd%&, BYVAL Zorder&, BYVAL X&, BYVAL Y&, BYVAL cx&, BYVAL cy&, BYVAL flag&)
END DECLARE

_TITLE "Windows Test"

'// Get the window handle (as a long)
hwnd%& = FindWindowA(0, "Windows Test" + CHR$(0))
PRINT "Handle:"; hwnd%&
_DELAY 3

'// Minimize the window - places program on the task bar
ret& = CloseWindow(hwnd%&)
PRINT "Minimize"; ret&
_DELAY 3 '// wait a few seconds

'// Restore the window
ret& = OpenIcon(hwnd%&)
PRINT "Restore"; ret&
_DELAY 3

'// Sets window priority as TOPMOST and moves the window position column 200, row 0
ret& = SetWindowPos(hwnd%&, HWND_TOPMOST, 200, 0, 0, 0, 0)
PRINT "Position"; ret&
_DELAY 3

'// Change the title of window header to new name
ret& = SetWindowTextA(hwnd%&, "Windows API Test")
PRINT "Title"; ret&

'// Get and display the process id for the window using new title name
PID& = GetWindowThreadProcessId(hwnd%&, 0)
PRINT "Process ID:"; PID&

END
```

Adapted from header code by Cyperium and Unseenmachine

Note: SetWindowPos& function sets the pixel location on the desktop and the window dimensions. It can also set the window's Z order priority.

The window handle value and process ID never change! Even when the title is changed.

(Return to Table of Contents)

Window Focus

Windows API routine can tell a program when it has lost focus by the user clicking on a different program window or the desktop.

```
DECLARE DYNAMIC LIBRARY "user32"
    FUNCTION GetForegroundWindow%& ()
    FUNCTION FindWindowA%& (BYVAL lpClassName%&, BYVAL lpWindowName%&)
END DECLARE

DIM title AS STRING
DIM hWnd AS _OFFSET

title = "Whatever you actually want the title to be"
_TITLE title
title = title + CHR$(0) 'always add character zero to FindWindowA string parameter
hWnd = FindWindowA(0, _OFFSET(title))

DO UNTIL LEN(INKEY$)
    IF hWnd = GetForegroundWindow THEN PRINT "foreground" ELSE PRINT "not foreground"
    SLEEP 1
LOOP
END
```

Adapted from code by Michael Calkins

Note: CHR$(0) could actually be added to the original title string and it wouldn't hurt anything. Compared values are _OFFSETs.

See SetForegroundWindow to set Focus on program.

(Return to Table of Contents)

Windows Menu

Creates a menu bar in the program window with a name that can be clicked on to execute a procedure.

```
DEFLNG A-Z

CONST MIIM_STATE = &H1
CONST MIIM_ID = &H2
CONST MIIM_TYPE = &H10
CONST MFT_SEPARATOR = &H800
CONST MFT_STRING = &H0
CONST MFS_ENABLED = &H0
CONST MFS_CHECKED = &H8

CONST HWND_TOPMOST = -1
CONST HWND_NOTOPMOST = -2
CONST SWP_NOMOVE = &H2
CONST SWP_NOSIZE = &H1
'--------------------------------------------------------------------------------

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 _OFFSET
    dwTypeData AS _OFFSET
    cch AS LONG
END TYPE

DECLARE LIBRARY
    FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$) ' To get hWnd handle
END DECLARE

DECLARE DYNAMIC LIBRARY "user32"
  FUNCTION CreateMenu& ()
  FUNCTION DrawMenuBar (BYVAL hWnd&)
  FUNCTION SetMenu& (BYVAL hWnd&, BYVAL hMenu&)
  FUNCTION InsertMenuItemA& (BYVAL hMenu&, BYVAL uItem&, BYVAL fByPosition&, BYVAL lpmii AS _OFFSET)
  FUNCTION GetMenuItemCount& (BYVAL hMenu&)
  FUNCTION GetMenuItemInfoA& (BYVAL hMenu&, BYVAL uItem&, BYVAL fByPosition&, BYVAL lpmii AS _OFFSET)
END DECLARE

DIM hWnd AS LONG
DIM hMenu AS LONG
DIM MenuItem AS MENUITEMINFO, BlankMenuItem AS MENUITEMINFO
DIM TypeData AS STRING * 1000

_TITLE "Menu bar API demo"
hWnd = FindWindow(0, "Menu bar API demo" + CHR$(0))

hMenu = CreateMenu: BlankMenuItem.cbSize = LEN(BlankMenuItem)

COLOR 7, 1: CLS
```

```
'Add a separator bar
count = GetMenuItemCount(hMenu): PRINT "MenuItemCount:"; count
MenuItem = BlankMenuItem
MenuItem.fMask = MIIM_ID OR MIIM_TYPE
MenuItem.fType = MFT_SEPARATOR
MenuItem.wID = count
IF InsertMenuItemA(hMenu, count, 1, _OFFSET(MenuItem)) THEN PRINT "Successfully added menu item!" ELSE PRINT "F

'Add a button
MenuItem = BlankMenuItem
count = GetMenuItemCount(hMenu): PRINT "MenuItemCount:"; count
MenuItem.fMask = MIIM_STATE OR MIIM_ID OR MIIM_TYPE
MenuItem.fType = MFT_STRING
MenuItem.fState = MFS_ENABLED
MenuItem.wID = count
TypeData = "&Fire Laser!" + CHR$(0)
MenuItem.dwTypeData = _OFFSET(TypeData)
MenuItem.cch = LEN(MenuItem.dwTypeData)
MyButton = MenuItem.wID
IF InsertMenuItemA(hMenu, count, 1, _OFFSET(MenuItem)) THEN PRINT "Successfully added menu item!" ELSE PRINT "F

IF SetMenu(hWnd, hMenu) THEN PRINT "Successfully set menu!": PRINT "Menu handle is:"; hMenu ELSE PRINT "Failed

DO: _LIMIT 70
    prev_state = new_state
    ok = GetMenuItemInfoA(hMenu, MyButton, 1, _OFFSET(MenuItem))
    new_state = MenuItem.fState AND 128
    IF prev_state = 0 AND new_state <> 0 THEN PRINT "Ouch! ";
LOOP WHILE INKEY$ = ""
```

Code by Galleon

Links/References: http://www.qb64.net/forum/index.php?topic=4735.msg48900#msg48900
http://www.ex-designz.net/apidetail.asp?api_id=168
http://msdn.microsoft.com/en-us/library/windows/desktop/ms647980%28v=vs.85%29.aspx

(Return to Table of Contents)

Windows Notification

The following code creates a pop-up notification balloon in the Windows task bar or from the Windows 10 side bar.

```
'public domain, 2012 feb, michael calkins

CONST NIM_ADD = 0
CONST NIM_MODIFY = 1
CONST NIM_DELETE = 2

CONST NIF_ICON = 2
CONST NIF_TIP = 4
CONST NIF_INFO = &H10

CONST NIIF_NONE = 0
CONST NIIF_INFO = 1
CONST NIIF_WARNING = 2
CONST NIIF_ERROR = 3
CONST NIIF_USER = 4

CONST IDI_APPLICATION = 32512
CONST IDI_HAND = 32513
CONST IDI_QUESTION = 32514
CONST IDI_EXCLAMATION = 32515
CONST IDI_ASTERISK = 32516

DECLARE DYNAMIC LIBRARY "kernel32"
 FUNCTION GetLastError~& ()
END DECLARE

DECLARE DYNAMIC LIBRARY "user32"
 FUNCTION FindWindowA%& (BYVAL lpClassName%&, BYVAL lpWindowName%&)
 FUNCTION LoadIconA%& (BYVAL hInstance%&, BYVAL lpIconName%&)
END DECLARE

DECLARE DYNAMIC LIBRARY "shell32"
 FUNCTION Shell_NotifyIconA& (BYVAL dwMessage~&, BYVAL lpdata%&)
END DECLARE

TYPE NOTIFYICONDATA
 cbSize AS _UNSIGNED LONG
 hWnd AS _OFFSET
 uID AS _UNSIGNED LONG
 uFlags AS _UNSIGNED LONG
 uCallbackMessage AS _UNSIGNED LONG
 hIcon AS _OFFSET
 szTip AS STRING * 128
 dwState AS _UNSIGNED LONG
 dwStateMask AS _UNSIGNED LONG
 szInfo AS STRING * 256
 uTimeout AS _UNSIGNED LONG
 szInfoTitle AS STRING * 64
 dwInfoFlags AS _UNSIGNED LONG
END TYPE
```

```
DIM hWnd AS _OFFSET
DIM hIcon AS _OFFSET
DIM t AS STRING
DIM notifydata AS NOTIFYICONDATA
notifydata.cbSize = LEN(notifydata)

t = "qb64 notification test"
_TITLE t
t = t + CHR$(0)
hWnd = FindWindowA(0, _OFFSET(t)) 'find window ID
IF hWnd = 0 THEN
 PRINT "FindWindowA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
 END
END IF

hIcon = LoadIconA(0, IDI_ASTERISK)
IF hIcon = 0 THEN
 PRINT "LoadIconA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
END IF
'first notification
notifydata.hWnd = hWnd
notifydata.uID = 0
notifydata.uFlags = NIF_ICON OR NIF_TIP OR NIF_INFO
notifydata.hIcon = hIcon
notifydata.szTip = "Connect charger!" + CHR$(0) 'tool tip
notifydata.szInfo = "Recharge" + CHR$(0) 'information
notifydata.uTimeout = 10000 'milliseconds
notifydata.szInfoTitle = "Low Battery" + CHR$(0) 'balloon title FALSE LOW BATTERY warning
notifydata.dwInfoFlags = NIIF_INFO

IF 0 = Shell_NotifyIconA(NIM_ADD, _OFFSET(notifydata)) THEN
 PRINT "Shell_NotifyIconA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
 END
END IF

PRINT "Press any key to modify it."
SLEEP: DO WHILE LEN(INKEY$): LOOP

hIcon = LoadIconA(0, IDI_HAND)
IF hIcon = 0 THEN
 PRINT "LoadIconA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
END IF
'second notification
notifydata.uFlags = NIF_ICON OR NIF_TIP OR NIF_INFO
notifydata.hIcon = hIcon
notifydata.szTip = "hahaha" + CHR$(0)
notifydata.szInfo = ":-)" + CHR$(0)
notifydata.uTimeout = 10000 'milliseconds
notifydata.szInfoTitle = "Howdy." + CHR$(0)
notifydata.dwInfoFlags = NIIF_WARNING

IF 0 = Shell_NotifyIconA(NIM_MODIFY, _OFFSET(notifydata)) THEN
 PRINT "Shell_NotifyIconA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
 END
END IF


PRINT "Press any key to delete the notification icon."
SLEEP: DO WHILE LEN(INKEY$): LOOP

IF 0 = Shell_NotifyIconA(NIM_DELETE, _OFFSET(notifydata)) THEN
 PRINT "Shell_NotifyIconA failed. Error: 0x" + LCASE$(HEX$(GetLastError))
 END
END IF

END
```

Adapted from code by Michael Calkins
NOTE: The program emulates a FALSE Low Battery warning!

MSDN References:

http://msdn.microsoft.com/en-us/library/windows/desktop/ms633499%28v=vs.85%29.aspx

http://msdn.microsoft.com/en-us/library/windows/desktop/ms648072%28v=vs.85%29.aspx

http://msdn.microsoft.com/en-us/library/windows/desktop/bb762159%28v=vs.85%29.aspx

http://msdn.microsoft.com/en-us/library/windows/desktop/bb773352%28v=vs.85%29.aspx

(Return to Table of Contents)

Windows Ports

The following library uses QueryDosDeviceA to find the COM or LPT ports on a Windows computer only.

```
'this example uses QueryDosDeviceA to enumerate COM ports.
'public domain, sept 2011, michael calkins
' http://www.qb64.net/forum/index.php?topic=4527.0
```

```
DECLARE DYNAMIC LIBRARY "kernel32"
 FUNCTION QueryDosDeviceA~& (BYVAL lpDeviceName AS _UNSIGNED _OFFSET, BYVAL lpTargetPath AS _UNSIGNED _OFFSET,
 FUNCTION GetLastError~& ()
END DECLARE

DIM sizeofbuffer AS _UNSIGNED LONG
DIM buffer AS STRING
DIM i AS _UNSIGNED LONG
DIM x AS _UNSIGNED LONG
DIM n AS _UNSIGNED LONG
sizeofbuffer = 1024
buffer = SPACE$(sizeofbuffer)

DO
 x = 0
 IF QueryDosDeviceA~&(0, _OFFSET(buffer), sizeofbuffer) = 0 THEN
  x = GetLastError~&
  IF x = &H7A THEN
   sizeofbuffer = sizeofbuffer + 1024
   buffer = SPACE$(sizeofbuffer)
  ELSE
   PRINT "Error: 0x"; HEX$(x)
   END
  END IF
 END IF
LOOP WHILE x = &H7A

i = 1
n = 0
DO WHILE ASC(MID$(buffer, i, 1))
 x = INSTR(i, buffer, CHR$(0))
 PRINT MID$(buffer, i, x - i)
 IF MID$(buffer, i, 3) = "COM" THEN 'change to "LPT" for parallel ports
  REDIM _PRESERVE comports(0 TO (n * 2) + 1) AS STRING
  comports(n * 2) = MID$(buffer, i, (x - i) + 1)
  n = n + 1
 END IF
 i = x + 1
LOOP

PRINT
PRINT n; "COM ports:"
IF n THEN
 FOR i = 0 TO n - 1
  DO
   x = 0
   IF QueryDosDeviceA~&(_OFFSET(comports(i * 2)), _OFFSET(buffer), sizeofbuffer) = 0 THEN
    x = GetLastError~&
    IF x = &H7A THEN
     sizeofbuffer = sizeofbuffer + 1024
     buffer = SPACE$(sizeofbuffer)
    ELSE
     PRINT "Error: 0x"; HEX$(x)
     END
    END IF
   END IF
  LOOP WHILE x = &H7A
  comports((i * 2) + 1) = LEFT$(buffer, INSTR(buffer, CHR$(0)) - 1)
  comports(i * 2) = LEFT$(comports(i * 2), LEN(comports(i * 2)) - 1)
  PRINT CHR$(&H22); comports(i * 2); CHR$(&H22); " is mapped to: "; CHR$(&H22); comports((i * 2) + 1); CHR$(&H2
 NEXT
END IF

buffer = ""

END
```

Code courtesy of Michael Calkins

(Return to Table of Contents)

Windows Sounds

MessageBeep plays Windows alert sound files located in the *C:\windows\media* folder. Some may not be set!

```
CONST MB_OK = 0 'beep
CONST MB_ICONERROR = &H10
CONST MB_ICONQUESTION = &H20
CONST MB_ICONWARNING = &H30
CONST MB_ICONASTERISK = &H40

DECLARE LIBRARY
  SUB MessageBeep (BYVAL alert AS _UNSIGNED LONG)
END DECLARE

PRINT "OK"
MessageBeep MB_OK

SLEEP 2
PRINT "Error"
MessageBeep MB_ICONERROR
```

```
SLEEP 2
PRINT "?"
MessageBeep MB_ICONQUESTION

SLEEP 2
PRINT "Warning"
MessageBeep MB_ICONWARNING

SLEEP 2
PRINT "Asterisk"
MessageBeep MB_ICONASTERISK
```

Code by Ted Weissgerber

Note: The sounds can be set in *Control Panel: Sounds and Audio Devices/Sounds* settings tab if not already assigned.

PlaySound plays Windows System sounds on most PC's:

```
'SDL-SPECIFIC CHANGES! GL only needs DECLARE LIBRARY without DLL name

DECLARE DYNAMIC LIBRARY "Winmm"
    FUNCTION PlaySound (pszSound AS STRING, BYVAL hmod AS INTEGER, BYVAL fdwSound AS INTEGER)
END DECLARE
CONST SND_ALIAS = 65536
CONST SND_ASYNC = 1

x = PlaySound("SystemExclamation" + CHR$(0), 0, SND_ALIAS + SND_ASYNC)
```

Use "SystemDefault", "SystemExclamation", "SystemExit", "SystemHand", "SystemQuestion", "SystemStart" or "SystemWelcome"

(Return to Table of Contents)

Window Transparency

Program changes transparency of the window. Plus key increases visibility while minus key can make it completely invisible.

```
DEFINT A-Z

' Declare windows API functions
DECLARE DYNAMIC LIBRARY "user32"
   FUNCTION SetLayeredWindowAttributes& (BYVAL hwnd AS LONG, BYVAL crKey AS LONG, BYVAL bAlpha AS _UNSIGNED _BY
   FUNCTION GetWindowLong& ALIAS "GetWindowLongA" (BYVAL hwnd AS LONG, BYVAL nIndex AS LONG)
   FUNCTION SetWindowLong& ALIAS "SetWindowLongA" (BYVAL hwnd AS LONG, BYVAL nIndex AS LONG, BYVAL dwNewLong AS
END DECLARE

' Needed for acquiring the hWnd of the window
DECLARE LIBRARY
   FUNCTION FindWindow& (BYVAL ClassName AS _OFFSET, WindowName$) ' To get hWnd handle
END DECLARE


DIM MyHwnd AS LONG

' Get hWnd value
_TITLE "Translucent window test"
MyHwnd = FindWindow(0, "Translucent window test" + CHR$(0))


' Set screen and draw a simple fractal which looks cooler than a black translucent window
SCREEN _NEWIMAGE(640, 480, 32)

FOR Py = 0 TO 479
   FOR Px = 0 TO 639
      PSET (Px, Py), _RGB32((Px OR Py) MOD 256, (Px + Py) MOD 256, Py MOD 256)
   NEXT Px
NEXT Py


' Main loop
PRINT "Press +/- to change opacity"

Level = 127
SetWindowOpacity MyHwnd, Level
DO
   Press$ = INKEY$
   LOCATE 2, 1: PRINT "Opacity:"; Level

   ' Change window opacity whenever +/- are pressed
   IF Press$ = "+" AND Level < 255 THEN Level = Level + 1: SetWindowOpacity MyHwnd, Level
   IF Press$ = "-" AND Level > 0 THEN Level = Level - 1: SetWindowOpacity MyHwnd, Level

   _LIMIT 60

LOOP UNTIL Press$ = CHR$(27)
SYSTEM


'=============================================================\
SUB SetWindowOpacity (hWnd AS LONG, Level)
DIM Msg AS LONG
```

```
CONST G = -20
CONST LWA_ALPHA = &H2
CONST WS_EX_LAYERED = &H80000

Msg = GetWindowLong(hWnd, G)
Msg = Msg OR WS_EX_LAYERED
Crap = SetWindowLong(hWnd, G, Msg)
Crap = SetLayeredWindowAttributes(hWnd, 0, Level, LWA_ALPHA)

END SUB
```

Code by Jobert

(Return to Table of Contents)

Windows User

Finds the current user's *My Documents* folder as a STRING value. Other environmental constants are listed below code.

```
'public domain

CONST MAX_PATH = 260              'maximum length of path string
CONST CSIDL_PERSONAL = &H5        'User's My Documents path. See environmental constants below
CONST SHGFP_TYPE_CURRENT = 0

CONST S_OK = 0                    'environmental value found
CONST S_FALSE = &H1               'folder does not exist
CONST E_FAILED = &H80004005&      'folder does not exist
CONST E_INVALIDARG = &H80070057& 'invalid argument parameter

DECLARE DYNAMIC LIBRARY "shell32"
 FUNCTION SHGetFolderPathA& (BYVAL hwndOwner%&, BYVAL nFolder&, BYVAL hToken%&, BYVAL dwFlags~&, BYVAL pszPath%
END DECLARE

DIM path AS STRING
DIM hr AS LONG
DIM n AS LONG

path = STRING$(MAX_PATH, 0) 'enlarge the string to MAX_PATH

hr = SHGetFolderPathA(0, CSIDL_PERSONAL, 0, SHGFP_TYPE_CURRENT, _OFFSET(path))
IF hr THEN
 PRINT "hresult: 0x" + LCASE$(HEX$(hr)) 'function returns non-zero error code
 END
END IF

n = INSTR(path, CHR$(0)) 'find terminating null

path = LEFT$(path, n - 1) 'shrink the string (probably creates a temp string)
PRINT CHR$(&H22); path; CHR$(&H22)
END
```

Code courtesy of Michael Calkins

Note: When the SHGetFolderPathA& function returns a non-zero value, the 0x hexadecimal error number is printed instead.

Vista and newer versions of Windows can also use SHGetKnownFolderPath and similar Constant values (http://msdn.microsoft.com/en-us/library/bb776911(v=vs.85))

Windows CSIDL Environmental LONG Constants

```
CONST CSIDL_DESKTOP = &H0                   '<user name>\desktop
CONST CSIDL_INTERNET = &H1                  'Internet Explorer(icon on desktop)
CONST CSIDL_PROGRAMS = &H2                  '<user name>\Start Menu\Programs
CONST CSIDL_CONTROLS = &H3                  'My Computer\Control Panel icon group
CONST CSIDL_PRINTERS = &H4                  'My Computer\Printers (installed)
CONST CSIDL_PERSONAL = &H5                  '<user name>\My Documents
CONST CSIDL_FAVORITES = &H6                 '<user name>\Favorites
CONST CSIDL_STARTUP = &H7                   '<user name>\Start Menu\Programs\Startup
CONST CSIDL_RECENT = &H8                    '<user name>\Recent
CONST CSIDL_SENDTO = &H9                    '<user name>\SendTo
CONST CSIDL_BITBUCKET = &HA                 '<user desktop>\Recycle Bin
CONST CSIDL_STARTMENU = &HB                 '<user name>\Start Menu
CONST CSIDL_MYDOCUMENTS = &HC               'logical "My Documents" desktop icon
CONST CSIDL_MYMUSIC = &HD                   '<user name>\My Documents\My Music folder
CONST CSIDL_MYVIDEO = &HE                   '<user name>\My Documents\My Videos folder
CONST CSIDL_DESKTOPDIRECTORY = &H10         '<user name>\Desktop
CONST CSIDL_DRIVES = &H11                   'My Computer
CONST CSIDL_NETWORK = &H12                  'Network Neighborhood (My Network Places)
CONST CSIDL_NETHOOD = &H13                  '<user name>\nethood
CONST CSIDL_FONTS = &H14                    'windows\fonts
CONST CSIDL_TEMPLATES = &H15                'templates
CONST CSIDL_COMMON_STARTMENU = &H16         'All Users\Start Menu
CONST CSIDL_COMMON_PROGRAMS = &H17          'All Users\Start Menu\Programs
CONST CSIDL_COMMON_STARTUP = &H18           'All Users\Startup
CONST CSIDL_COMMON_DESKTOPDIRECTORY = &H19 'All Users\Startup
CONST CSIDL_APPDATA = &H1A                  '<user name>\Application Data
CONST CSIDL_PRINTHOOD = &H1B                '<user name>\PrintHood
CONST CSIDL_LOCAL_APPDATA = &H1C            '<user name>\Local Settings\Application Data
CONST CSIDL_ALTSTARTUP = &H1D               'non-localized startup(Vista only)
CONST CSIDL_COMMON_ALTSTARTUP = &H1E        'common startup
CONST CSIDL_COMMON_FAVORITES = &H1F         'common startup
```

```
CONST CSIDL_INTERNET_CACHE = &H20          'common startup
CONST CSIDL_COOKIES = &H21                 'common startup
CONST CSIDL_HISTORY = &H22                 'common startup
CONST CSIDL_COMMON_APPDATA = &H23          'All Users\Application Data
CONST CSIDL_WINDOWS = &H24                 'GetWindowsDirectory()
CONST CSIDL_SYSTEM = &H25                  'GetSystemDirectory()
CONST CSIDL_PROGRAM_FILES = &H26           'C:\Program Files
CONST CSIDL_MYPICTURES = &H27              '<user name>\My Documents\My Pictures
CONST CSIDL_PROFILE = &H28                 'USERPROFILE
CONST CSIDL_SYSTEMX86 = &H29               'x86 C:\Windows\System32 or SysWOW64(64 bit PC)
CONST CSIDL_PROGRAM_FILESX86 = &H2A        'x86 C:\Program Files on RISC
CONST CSIDL_PROGRAM_FILES_COMMON = &H2B    'C:\Program Files\Common
CONST CSIDL_PROGRAM_FILES_COMMONX86 = &H2C 'x86 Program Files\Common on RISC
CONST CSIDL_COMMON_TEMPLATES = &H2D        'All Users\Templates
CONST CSIDL_COMMON_DOCUMENTS = &H2E        'All Users\Documents
CONST CSIDL_COMMON_ADMINTOOLS = &H2F       'All Users\Start Menu\Programs\Administrative Tools
CONST CSIDL_ADMINTOOLS = &H30              '<user name>\Start Menu\Programs\Administrative Tools
CONST CSIDL_CONNECTIONS = &H31             'Network and Dial-up Connections
CONST CSIDL_COMMON_MUSIC = &H35            'All Users\My Music
CONST CSIDL_COMMON_PICTURES = &H36         'All Users\My Pictures
CONST CSIDL_COMMON_VIDEO = &H37            'All Users\My Video
CONST CSIDL_RESOURCES = &H38               'Resource Directory
CONST CSIDL_RESOURCES_LOCALIZED = &H39     'Localized Resource Directory
CONST CSIDL_COMMON_OEM_LINKS = &H3A        'Links to All Users OEM specific apps
CONST CSIDL_CDBURN_AREA = &H3B             'USERPROFILE\Local Settings\Application Data\Microsoft\CD Burning

                'See: http://msdn.microsoft.com/en-us/library/bb762181%28v=vs.85%29
```

(Return to Table of Contents)

Windows Version

Note that QB64 will not run on Windows 95 to ME computers currently!

```
DECLARE LIBRARY
  FUNCTION GetVersion ()
END DECLARE

'Just grab the "build" number...
b$ = LTRIM$(RTRIM$(STR$((GetVersion AND &HFFFF0000) \ &H10000)))


PRINT "The Windows version is: ";

IF INSTR(1, b$, "095") THEN PRINT "Windows 95"
IF INSTR(1, b$, "1111") THEN PRINT "Windows 95"
IF INSTR(1, b$, "1381") THEN PRINT "Windows NT"
IF INSTR(1, b$, "1998") THEN PRINT "Windows 98"
IF INSTR(1, b$, "2222") THEN PRINT "Windows 98 SE"
IF INSTR(1, b$, "3000") THEN PRINT "Windows ME"
IF INSTR(1, b$, "2195") THEN PRINT "Windows 2000"
IF INSTR(1, b$, "2600") THEN PRINT "Windows XP"
IF INSTR(1, b$, "3790") THEN PRINT "Windows Server 2003"
IF INSTR(1, b$, "6000") THEN PRINT "Windows Vista/Server"
IF INSTR(1, b$, "6001") THEN PRINT "Windows Vista/Server"
IF INSTR(1, b$, "6002") THEN PRINT "Windows Vista/Server"
IF INSTR(1, b$, "7600") THEN PRINT "Windows 7"
```

Code courtesy of Dav

(Return to Table of Contents)

Reference

Note: C++ Header files should be placed in the QB64 folder and are not required after a program is compiled.

Your code contribution using the Windows Libraries could end up here!

See also:

- _WINDOWHANDLE
- Windows Environment
- DECLARE LIBRARY, BYVAL, ALIAS
- _OFFSET, _OFFSET (function) (lp, ptr and p names)
- SDL Libraries, DLL Libraries, C Libraries
- C++ Variable Types
- VB Script
- Windows Printer Settings

Navigation:

Go to Keyword Reference - Alphabetical
Go to Keyword Reference - By usage
Go to Main WIKI Page