VB Code Help File **version 1.1**

Help file written by Jeff Williams

For Help on Help, Press F1

Cosmetics

Centerform

Centers a form on the screen

FormStayOnTop

FormStayOnTop establishes the specified window as the topmost window no matter which window is active.

Pass it the handle of the window you want to make topmost (or for which you wish to end that condition) and a true/false flag to indicate whether it should be topmost.

Shadow

This subroutine draws a gray shadow below and to the right of a control.
NOTE: This subroutine must be in your form.

TilePaper

This function will tile a picture on a form. Call this procedure in the Paint and Resize procedures of the form.

Dialogs

AlertBox

Creates an Alert Box using specified text and App.Title

Confirm

Function creates confirmation box using specified text, returns True if Yes button pressed, False if No button pressed

File

CountSubDirs

This is a function that counts subdirectorys immediately under that directory

CreatePath

Creates the passed path

Exists

The Exists%() function returns a value of TRUE if the specified file exists, or FALSE if it doesn't.

FixPath

Function returns a passed path with backslash at end.

GetLoadedFilePath

Find the path of a loaded file (for example: the applications own path)

Instances

Returns the number of instances a program is running

IsValidPath

Function determines if passed pathname is valid

OpenFileDialog

Brings up the Open File (Browse) dialog from COMMDLG.DLL
OpenCommDlg (Filter, Extension, Directory, Title)

StripPath

Function removes path from fully-qualified file name, returns file name only

Groups and Icons

CreateProgManGroup

Creates a Program Manager group

CreateProgManItem

Creates a program manager item (icon)

IconExtractor

Extracts icons from a specified Exe file

Information

GetDOSVersion

Get the current DOS version

GetIni

Read an entry in an INI file

GetSysDir

Returns the current Windows SYSTEM directory

GetWinDir

Return the Windows directory

GetWindowsVersion

- The Windows version number with GetVersion
- The kind of CPU (80286, 80386, or 80486) and whether a math coprocessor is present with GetWinFlags
- Whether Windows is running in enhanced mode or standard mode with GetWinFlags
- The amount of free memory with GetFreeSpace and GlobalCompact
- The percentage of free system resources with SystemHeapInfo

PutIni

Writes an entry to an INI file

Registration Database

Registration database functions (AddKey, EnumRoot, GetCurVer, GetKey, GetServer)

Mail

SendMail

Sends a mail message using the MAPI DLL

Math

CheckPeriod

This subroutine makes sure your text box never has more than one period in it

Default to Integer

This command causes all undeclared variables to default to integers (the fastest variable type)

HighWord

Obtain hiword of long

LoWord

Obtain LoWord of Long

NumbersOnly

Only allow numbers in a text box (requires CheckPeriod)

Miscellaneous

FixAPI

Trims spaces CHR$(0)'s from string returned by API function

Network

GetWorkStationInfo

Get machine network information like the computer, user name, workgroup and domain

Net Connections

Connect or Disconnect from a Network Path

Sound

MakeBeep

MakeBeep beeps the PC's speaker a specified number of times

PlayWaveFile

Play a Wave file

Time

Elapse

This is a timer function

WaitSecs

Wait a number of seconds

Window Handles

FindParent

Routine to locate the progenitor of a series of Windows

FindAndRestorePrevInstance

Finds and restores a previous running instance of your app

Loaded

Loaded tells if an app of the passed classname is loaded

RestoreApp

RestoreApp restores the windows whose handle you pass to it

SearchWindowList

Function returns handle of first window matching partial name parameter

TrackPopupMenu

Tracks a popup menu.

Pass it the number (going from right to left) of the menu you wish to view, the X & Y coordinates at which it should pop up (as returned by a mousedown event), the form on which the mousedown event took place (and over which the menu should appear), and the form to which the menu belongs (which may or may not be the same as the previous form).

Centerform

```
Sub CenterForm (aForm As Form)

    aForm.Move (Screen.Width - aForm.Width) / 2, (Screen.Height -
aForm.Height) / 2

End Sub
```

FormStayOnTop

```
'Declares for FormStayOnTop

Declare Sub SetWindowPos Lib "User" (ByVal hWnd As Integer, ByVal
hWndInsertAfter As Integer, ByVal X As Integer, ByVal Y As Integer, ByVal cx
As Integer, ByVal cy As Integer, ByVal wFlags As Integer)


Sub FormStayOnTop (varForm as Form, OnTop%)


Handle% = varForm.hwnd


    Const Swp_Nosize = &H1

    Const SWP_Nomove = &H2

    Const Swp_NoActivate = &H10

    Const Swp_ShowWindow = &H40

    Const Hwnd_TopMost = -1

    Const Hwnd_NoTopMost = -2

    wFlags = SWP_Nomove Or Swp_Nosize Or Swp_ShowWindow Or Swp_NoActivate


    Select Case OnTop%

         Case True

            PosFlag = Hwnd_TopMost

         Case False

             PosFlag = Hwnd_NoTopMost

         End Select

    SetWindowPos Handle%, PosFlag, 0, 0, 0, 0, wFlags


End Sub
```

Shadow

```
Sub Shadow (f As Form, c As Control)
    Const color = &HC0C0C0      ' Color of the shadow
    Const shWidth = 3           ' Width of the shadow
    Dim oldWidth As Integer     ' Saves old DrawWidth
    Dim oldScale As Integer     ' Saves old ScaleMode

    oldWidth = f.DrawWidth      ' Remember current DrawWidth
    oldScale = f.ScaleMode      ' Remember current ScaleMode

    f.ScaleMode = 3             ' Set to Pixel scaling
    f.DrawWidth = 1             ' 1-pixel wide lines
    '
    ' Draws the shadow around the control by drawing a gray
    ' box behind the control that's offset right and down.
    '
    f.Line (c.Left + shWidth, c.Top + shWidth)-Step(c.Width - 1,
c.Height - 1), color, BF

    f.DrawWidth = oldWidth      ' Restore old DrawWidth
    f.ScaleMode = oldScale      ' Restore old ScaleMode
End Sub
```

TilePaper

```
'Declares for TilePaper

Declare Function BitBlt% Lib "GDI" (ByVal hdestDC As Integer, ByVal x As
Integer, ByVal y As Integer, ByVal nWidth As Integer, ByVal nHeight As
Integer, ByVal hsrcDC As Integer, ByVal xsrc As Integer, ByVal ysrc As
Integer, ByVal dwRop As Long)

Const copyit& = &HCC0020


Function Tilepaper (Pic As Control, Frm As Form)

'Pic is the picture box and Frm is the form it's on

'Call this procedure in the Paint and Resize procedures of the form


Pic.Borderstyle = 0

Frm.AutoRedraw = False

ScaleMode = 3

Wide% = Pic.Width

High% = Pic.Height

For y = 0 To Frm.ScaleHeight Step High%

    For x = 0 To Frm.ScaleWidth Step Wide%

        z% = BitBlt(Frm.hDC, x, y, Wide%, High%, Pic.hDC, 0, 0, copyit&)

    Next x

Next y

Tilepaper = True

End Function
```

AlertBox

```
Sub Alert (Mess$)
    '  * creates an Alert box with an OK button
    MsgBox Mess$, 48, App.Title
End Sub
```

Confirm

```
Function Confirm% (Ask$)
    If MsgBox(Ask$, 52, App.Title) = 6 Then Confirm% = True
End Function
```

CountSubDirs

```
Function CountSubDirs (vDir As String) As Integer
If Right(vDir, 1) <> "\" Then vDir = vDir & "\"
x$ = Dir$(vDir, 16)
If x$ = "" Then
    CountSubDirs = 0
    Exit Function
End If
Do While x$ <> ""
    If Trim(x$) <> "." And Trim(x$) <> ".." Then
        If GetAttr(vDir + x$) >= 16 And GetAttr(vDir + x$) < 32 Then
            num% = num% + 1
        End If
    End If
    x$ = Dir$
Loop
CountSubDirs = num%
End Function
```

CreatePath

```
Function CreatePath (ByVal DestPath$) As Integer
'-------------------------------------------
' Create the path contained in DestPath$
' First char must be drive letter, followed by
' a ":\" followed by the path, if any.
'-------------------------------------------

    Screen.MousePointer = 11

    '-------------------------------------------
    ' Add slash to end of path if not there already
    '-------------------------------------------
    If Right$(DestPath$, 1) <> "\" Then
        DestPath$ = DestPath$ + "\"
    End If



    '----------------------------------
    ' Change to the root dir of the drive
    '----------------------------------
    On Error Resume Next
    ChDrive DestPath$
    If Err <> 0 Then GoTo errorOut
    ChDir "\"

    '-------------------------------------------------
    ' Attempt to make each directory, then change to it
    '-------------------------------------------------
    BackPos = 3
    forePos = InStr(4, DestPath$, "\")
    Do While forePos <> 0
```

```
        temp$ = Mid$(DestPath$, BackPos + 1, forePos - BackPos - 1)

        Err = 0
        MkDir temp$
        If Err <> 0 And Err <> 75 Then GoTo errorOut

        Err = 0
        ChDir temp$
        If Err <> 0 Then GoTo errorOut

        BackPos = forePos
        forePos = InStr(BackPos + 1, DestPath$, "\")
    Loop

    CreatePath = True
    Screen.MousePointer = 0
    Exit Function

errorOut:
    MsgBox "Error While Attempting to Create Directories on Destination
Drive.", 48, "SETUP"
    CreatePath = False
    Screen.MousePointer = 0

End Function
```

Exists

```
Function Exists% (F$)
    On Error Resume Next
    X& = FileLen(F$)
    If X& Then Exists% = True
End Function
```

FixPath

```
Function FixPath$ (Test$)
    'sticks a backslash on the end of test$ if there's
    'not one there already
    Dim T$
    T$ = Test$
    If Right$(T$, 1) <> "\" Then T$ = T$ + "\"
   FixPath$ = T$
End Function
```

GetLoadedFilePath

```
'Declares for GetLoadedFilePath

Declare Function GetModuleHandle Lib "Kernel" (ByVal FileName$) As Integer

Declare Function GetModuleFileName Lib "Kernel" (ByVal hModule%, ByVal
FileName$, ByVal nsize%) As Integer


Function GetLoadedFilePath (FileName As String) As String

    Dim hmod As Integer, Path As String, PathLen As Integer

    hmod = GetModuleHandle%(FileName)

    Path = Space(255)

    PathLen = GetModuleFileName%(hmod, Path, Len(Path))

    GetLoadedFilePath = Left(Path, PathLen)

End Function
```

Instances

```
'Declares for Instances
Declare Function GetModuleHandle Lib "Kernel" (ByVal FileName$) As Integer
Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule%) As Integer

Function Instances (FileName As String) As Integer
    Instances = GetModuleUsage(GetModuleHandle(FileName))
End Function
```

IsValidPath

```
'-------------------------------------------------------
' Function:   IsValidPath as integer
' arguments:  DestPath$         a string that is a full path
'             DefaultDrive$     the default drive.  eg.  "C:"
'
'  If DestPath$ does not include a drive specification,
'  IsValidPath uses Default Drive
'
'  When IsValidPath is finished, DestPath$ is reformated
'  to the format "X:\dir\dir\dir\"
'
' Result:  True (-1) if path is valid.
'          False (0) if path is invalid
'-------------------------------------------------------
Function IsValidPath (DestPath$, ByVal DefaultDrive$) As Integer


    '-----------------------------
    ' Remove left and right spaces
    '-----------------------------
    DestPath$ = RTrim$(LTrim$(DestPath$))



    '-----------------------------
    ' Check Default Drive Parameter
    '-----------------------------
    If Right$(DefaultDrive$, 1) <> ":" Or Len(DefaultDrive$) <> 2 Then
        MsgBox "Bad default drive parameter specified in IsValidPath
Function.  You passed,  """ + DefaultDrive$ + """.  Must be one drive letter
and "":"".  For example, ""C:"", ""D:""...", 64, "Setup Kit Error"
        GoTo parseErr
    End If
```

```
'-----------------------------------------------------
' Insert default drive if path begins with root backslash
'-----------------------------------------------------
If Left$(DestPath$, 1) = "\" Then
    DestPath$ = DefaultDrive + DestPath$
End If

'-----------------------------
' check for invalid characters
'-----------------------------
On Error Resume Next
tmp$ = Dir$(DestPath$)
If Err <> 0 Then
    GoTo parseErr
End If


'----------------------------------------
' Check for wildcard characters and spaces
'----------------------------------------
If (InStr(DestPath$, "*") <> 0) GoTo parseErr
If (InStr(DestPath$, "?") <> 0) GoTo parseErr
If (InStr(DestPath$, " ") <> 0) GoTo parseErr


'----------------------------------------
' Make Sure colon is in second char position
'----------------------------------------
If Mid$(DestPath$, 2, 1) <> Chr$(58) Then GoTo parseErr


'-------------------------------
```

```
    ' Insert root backslash if needed
    '------------------------------
    If Len(DestPath$) > 2 Then
      If Right$(Left$(DestPath$, 3), 1) <> "\" Then
        DestPath$ = Left$(DestPath$, 2) + "\" + Right$(DestPath$,
Len(DestPath$) - 2)
      End If
    End If


    '-------------------------
    ' Check drive to install on
    '-------------------------
    drive$ = Left$(DestPath$, 1)
    ChDrive (drive$)                                                    '
Try to change to the dest drive
    If Err <> 0 Then GoTo parseErr


    '-----------
    ' Add final \
    '-----------
    If Right$(DestPath$, 1) <> "\" Then
        DestPath$ = DestPath$ + "\"
    End If



    '-----------------------------------
    ' Root dir is a valid dir
    '-----------------------------------
    If Len(DestPath$) = 3 Then
        If Right$(DestPath$, 2) = ":\" Then
            GoTo ParseOK
        End If
    End If
```

```
    '------------------------
    ' Check for repeated Slash
    '------------------------
    If InStr(DestPath$, "\\") <> 0 Then GoTo parseErr

    '--------------------------------------
    ' Check for illegal directory names
    '--------------------------------------
    legalChar$ = "!#$%&'()-0123456789@ABCDEFGHIJKLMNOPQRSTUVWXYZ^_`{}~."
    BackPos = 3
    forePos = InStr(4, DestPath$, "\")
    Do
        temp$ = Mid$(DestPath$, BackPos + 1, forePos - BackPos - 1)

        '----------------------------
        ' Test for illegal characters
        '----------------------------
        For i = 1 To Len(temp$)
            If InStr(legalChar$, UCase$(Mid$(temp$, i, 1))) = 0 Then GoTo
parseErr
        Next i

        '-----------------------------------------
        ' Check combinations of periods and lengths
        '-----------------------------------------
        periodPos = InStr(temp$, ".")
        length = Len(temp$)
        If periodPos = 0 Then
            If length > 8 Then GoTo parseErr                          ' Base
too long
        Else
```

```
            If periodPos > 9 Then GoTo parseErr                         ' Base
too long
            If length > periodPos + 3 Then GoTo parseErr                '
Extension too long
            If InStr(periodPos + 1, temp$, ".") <> 0 Then GoTo parseErr' Two
periods not allowed
        End If


        BackPos = forePos
        forePos = InStr(BackPos + 1, DestPath$, "\")
    Loop Until forePos = 0


ParseOK:
    IsValidPath = True
    Exit Function


parseErr:
    IsValidPath = False
End Function
```

OpenFileDialog

```
'Declares for Open File Dialog

Type tagOPENFILENAME

     lStructSize As Long

     hwndOwner As Integer

     hInstance As Integer

     lpstrFilter As Long

     lpstrCustomFilter As Long

     nMaxCustFilter As Long

     nFilterIndex As Long

     lpstrFile As Long

     nMaxFile As Long

     lpstrFileTitle As Long

     nMaxFileTitle As Long

     lpstrInitialDir As Long

     lpstrTitle As Long

     Flags As Long

     nFileOffset As Integer

     nFileExtension As Integer

     lpstrDefExt As Long

     lCustData As Long

     lpfnHook As Long

     lpTemplateName As Long

End Type


Declare Function GetOpenFileName% Lib "COMMDLG.DLL" (OPENFILENAME As
tagOPENFILENAME)

Declare Function GetSaveFileName% Lib "COMMDLG.DLL" (OPENFILENAME As
tagOPENFILENAME)

Declare Function lstrcpy& Lib "Kernel" (ByVal lpDestString As Any, ByVal
lpSourceString As Any)


Dim OPENFILENAME As tagOPENFILENAME
```

```
Global Const OFN_READONLY = &H1
Global Const OFN_OVERWRITEPROMPT = &H2
Global Const OFN_HIDEREADONLY = &H4
Global Const OFN_NOCHANGEDIR = &H8
Global Const OFN_SHOWHELP = &H10
Global Const OFN_ENABLEHOOK = &H20
Global Const OFN_ENABLETEMPLATE = &H40
Global Const OFN_ENABLETEMPLATEHANDLE = &H80
Global Const OFN_NOVALIDATE = &H100
Global Const OFN_ALLOWMULTISELECT = &H200
Global Const OFN_EXTENSIONDIFFERENT = &H400
Global Const OFN_PATHMUSTEXIST = &H800
Global Const OFN_FILEMUSTEXIST = &H1000
Global Const OFN_CREATEPROMPT = &H2000
Global Const OFN_SHAREAWARE = &H4000
Global Const OFN_NOREADONLYRETURN = &H8000
Global Const OFN_NOTESTFILECREATE = &H10000

Global Const OFN_SHAREFALLTHROUGH = 2
Global Const OFN_SHARENOWARN = 1
Global Const OFN_SHAREWARN = 0

Function OpenCommDlg (Filter$, DefExt$, szCurDir$, Title$)

' Filter$ is the dropdown of file types (leave blank for defaults)
' DefExt$ is used if the user does not specify an extension
' szCurDir$ is the default directory
' Title$ is the dialog title

Dim Message$, FileName$, FileTitle$, APIResults%

    If Filter$ = "" Then
```

```
        Filter$ = "All Files(*.*)" & Chr$(0) & "*.*" & Chr$(0)
        Filter$ = Filter$ & "Text(*.txt)" & Chr$(0) & "*.TXT" & Chr$(0)
        Filter$ = Filter$ & "Program Files" & Chr$(0) &
"*.EXE;*.COM;*.BAT;*.PIF" & Chr$(0)
        Filter$ = Filter$ & Chr$(0)
    End If

    '* Allocate string space for the returned strings.
    FileName$ = Chr$(0) & Space$(255) & Chr$(0)
    FileTitle$ = Space$(255) & Chr$(0)

    '* Give the dialog a caption title.
    Title$ = Title$ & Chr$(0)

    '* If the user does not specify an extension, append TXT.
    DefExt$ = DefExt$ & Chr$(0)

    '* Set up the defualt directory
    szCurDir$ = CurDir$ & Chr$(0)

    '* Set up the data structure before you call the GetOpenFileName

    OPENFILENAME.lStructSize = Len(OPENFILENAME)

    'If the OpenFile Dialog box is linked to a form use this line.
    'It will pass the forms window handle.

    OPENFILENAME.hwndOwner = Screen.ActiveForm.hWnd

    'If the OpenFile Dialog box is not linked to any form use this line.
    'It will pass a null pointer.

    OPENFILENAME.hwndOwner = 0&
```

```
OPENFILENAME.lpstrFilter = lstrcpy(Filter$, Filter$)
OPENFILENAME.nFilterIndex = 1
OPENFILENAME.lpstrFile = lstrcpy(FileName$, FileName$)
OPENFILENAME.nMaxFile = Len(FileName$)
OPENFILENAME.lpstrFileTitle = lstrcpy(FileTitle$, FileTitle$)
OPENFILENAME.nMaxFileTitle = Len(FileTitle$)
OPENFILENAME.lpstrTitle = lstrcpy(Title$, Title$)
OPENFILENAME.Flags = 0 ' OFN_FILEMUSTEXIST Or OFN_READONLY
OPENFILENAME.lpstrDefExt = lstrcpy(DefExt$, DefExt$)
OPENFILENAME.hInstance = 0
OPENFILENAME.lpstrCustomFilter = 0
OPENFILENAME.nMaxCustFilter = 0
OPENFILENAME.lpstrInitialDir = lstrcpy(szCurDir$, szCurDir$)
OPENFILENAME.nFileOffset = 0
OPENFILENAME.nFileExtension = 0
OPENFILENAME.lCustData = 0
OPENFILENAME.lpfnHook = 0
OPENFILENAME.lpTemplateName = 0

'* This will pass the desired data structure to the Windows API,
'* which will in turn it uses to display the Open Dialog form.

APIResults% = GetOpenFileName(OPENFILENAME)

If APIResults% <> 0 Then
    '* Note that FileName$ will have an embedded Chr$(0) at the
    '* end. You may wish to strip this character from the string.
  FileName$ = Left$(FileName$, InStr(FileName$, Chr$(0)) - 1)
  OpenCommDlg = FileName$
Else
  OpenCommDlg = ""
End If
```

```
End Function
```

StripPath

```
Function StripPath$ (T$)
    Dim x%, ct%
    StripPath$ = T$
    x% = InStr(T$, "\")
    Do While x%
        ct% = x%
        x% = InStr(ct% + 1, T$, "\")
    Loop
    If ct% > 0 Then StripPath$ = Mid$(T$, ct% + 1)
End Function
```

CreateProgManGroup

```
Sub CreateProgManGroup (x As Form, GroupName$, GroupPath$)
'------------------------------------------------------------
' Procedure: CreateProgManGroup
' Arguments: X           The Form where a Label1 exist
'            GroupName$  A string that contains the group name
'            GroupPath$  A string that contains the group file
'                        name  ie 'myapp.grp'
'------------------------------------------------------------


    Screen.MousePointer = 11


    '--------------------------------------------------------------------
    ' Windows requires DDE in order to create a program group and item.
    ' Here, a Visual Basic label control is used to generate the DDE messages
    '--------------------------------------------------------------------
    On Error Resume Next



    '--------------------------------
    ' Set LinkTopic to PROGRAM MANAGER
    '--------------------------------
    x.Label1.LinkTopic = "ProgMan|Progman"
    x.Label1.LinkMode = 2
    For i% = 1 To 10                                        ' Loop to ensure
that there is enough time to
      z% = DoEvents()                                       ' process DDE
Execute.  This is redundant but needed
    Next                                                    ' for debug
windows.
    x.Label1.LinkTimeout = 100
```

```
    '---------------------
    ' Create program group
    '---------------------
    x.Label1.LinkExecute "[CreateGroup(" + GroupName$ + Chr$(44) + GroupPath$
+ ")]"



    '-----------------
    ' Reset properties
    '-----------------
    x.Label1.LinkTimeout = 50
    x.Label1.LinkMode = 0


    Screen.MousePointer = 0
End Sub
```

CreateProgManItem

```
Sub CreateProgManItem (x As Form, CmdLine$, IconTitle$)

'------------------------------------------------------
' Procedure: CreateProgManItem
'
' Arguments: X           The form where Label1 exists
'
'            CmdLine$    A string that contains the command
'                        line for the item/icon.
'                        ie 'c:\myapp\setup.exe'
'
'            IconTitle$  A string that contains the item's
'                        caption
'------------------------------------------------------

    Screen.MousePointer = 11

    '-----------------------------------------------------------------------
    ' Windows requires DDE in order to create a program group and item.
    ' Here, a Visual Basic label control is used to generate the DDE messages
    '-----------------------------------------------------------------------
    On Error Resume Next



    '---------------------------------
    ' Set LinkTopic to PROGRAM MANAGER
    '---------------------------------
    x.Label1.LinkTopic = "ProgMan|Progman"
    x.Label1.LinkMode = 2
    For i% = 1 To 10                                        ' Loop to ensure
that there is enough time to
      z% = DoEvents()                                       ' process DDE
```

```
Execute.  This is redundant but needed
    Next                                                    ' for debug
windows.
    x.Label1.LinkTimeout = 100



    '-----------------------------------------------
    ' Create Program Item, one of the icons to launch
    ' an application from Program Manager
    '-----------------------------------------------
    x.Label1.LinkExecute "[AddItem(" + CmdLine$ + Chr$(44) + IconTitle$ +
Chr$(44) + ",,)]"


    '-----------------
    ' Reset properties
    '-----------------
    x.Label1.LinkTimeout = 50
    x.Label1.LinkMode = 0


    Screen.MousePointer = 0
End Sub
```

IconExtractor

```
'Declares for IconExtractor

Const GWW_HINSTANCE = (-6)

Declare Function GetWindowWord Lib "User" (ByVal hWnd As Integer, ByVal
nIndex As Integer) As Integer

Declare Function ExtractIcon Lib "shell" (ByVal lpHandle As Integer, ByVal
lpExe As String, ByVal lpiconindex As Integer) As Integer

Declare Function DrawIcon Lib "USER" (ByVal lpHandle As Integer, ByVal xcoord
As Integer, ByVal ycoord As Integer, ByVal Hicon As Integer) As Integer


Sub IconExtractor (ExeFile$, F as Form, Pic as Picture)

Handle = F.hWnd

z = SCREEN.HEIGHT

    Select Case z

        Case 7000

            X = 2: Y = 1

        Case 7200

            X = 3: Y = 0

        Case 9000

            X = 3: Y = 0

        Case Is > 9000

            X = 8: Y = 4

    End Select


    Static Looper

    Looper = Looper + 1

    Inst = GetWindowWord(Handle, GWW_HINSTANCE)

    Hicon = ExtractIcon(Inst, ExeFile$, Looper - 1)

    If Hicon = 0 Then

        If Looper > 0 Then

            Hicon = ExtractIcon(Inst, ExeFile$, 0)

            Looper = 1

        Else Beep: Exit Sub
```

```
        End If
    End If
    F.Pic.CLS
    Draw = DrawIcon(F.Pic.hDC, X, Y, Hicon)
End Sub
```

GetDOSVersion

```
'Declares for GetDOSVersion
Declare Function GetVersion Lib "Kernel" () As Long


Function GetDOSVersion () As Single
    '
    ' This function returns the DOS version number as a
    ' floating point number, such as 3.31 for DOS 3.31.
    '
    Dim ver As Long
    Dim DOSVer As Single

    ver = GetVersion()              ' Get DOS/Windows version
    ver = ver / 65536               ' Get just the DOS version
    DOSVer = ver / 256              ' Get major version number
    DOSVer = DOSVer + (ver Mod 256) / 100
    GetDOSVersion = DOSVer             ' Return version
End Function
```

GetIni

```
'Declares for GetIni

Declare Function GetPrivateProfileString Lib "Kernel" (ByVal
lpApplicationName As String, ByVal lpKeyName As Any, ByVal lpDefault As
String, ByVal lpReturnedString As String, ByVal nsize As Integer, ByVal
lpFileName As String) As Integer


Function GetIni (FileName As String, Section As String, Keyword As String)

    Dim msg, success, x As String

    Dim Result As String * 128

    success = GetPrivateProfileString(Section, Keyword, "", Result,
Len(Result), FileName)

    If Left$(Result, 1) <> Chr$(0) Then

        x = Left$(Result, InStr(Result, Chr$(0)) - 1)

    Else

        x = ""

    End If

    GetIni = UCase(x)

End Function
```

GetSysDir

```
'Declares for GetSysDir

Declare Function GetSystemDirectory Lib "Kernel" (ByVal P$, ByVal S%) As
Integer


Function GetSysDir () As String

    Dim Gwdvar As String, Gwdvar_Length As Integer

    Gwdvar = Space(255)

    Gwdvar_Length = GetSystemDirectory(Gwdvar, 255)

    VB_SysDir = Left(Gwdvar, Gwdvar_Length)

End Function
```

GetWinDir

```
'Declares for GetWinDir

Declare Function GetWindowsDirectory Lib "KERNEL" (ByVal lpbuffer As String,
ByVal nsize As Integer) As Integer



Function VB_WinDir () As String

    Dim Gwdvar As String, Gwdvar_Length As Integer

    Gwdvar = Space(255)

    Gwdvar_Length = GetWindowsDirectory(Gwdvar, 255)

    VB_WinDir = Left(Gwdvar, Gwdvar_Length)

End Function
```

Registration Database

```
'Declares for Registration Database

Declare Function RegOpenKey& Lib "SHELL.DLL" (ByVal hKey&, ByVal lpszSubKey$,
lphKey&)

Declare Function RegCreateKey& Lib "SHELL.DLL" (ByVal hKey&, ByVal
lpszSubKey$, lphKey&)

Declare Function RegQueryValue& Lib "SHELL.DLL" (ByVal hKey&, ByVal
lpszSubKey$, ByVal lpszValue$, nSize&)

Declare Function RegEnumKey& Lib "SHELL.DLL" (ByVal hKey&, ByVal iSubKey&,
ByVal lpReturnedString$, ByVal nSize&)

Declare Function RegSetValue& Lib "SHELL.DLL" (ByVal hKey&, ByVal
lpszSubKey$, ByVal fdwType&, ByVal lpszValue$, ByVal dwLength&)

Declare Function RegDeleteKey& Lib "SHELL.DLL" (ByVal hKey&, ByVal
lpszSubKey$)

Declare Function RegCloseKey& Lib "SHELL.DLL" (ByVal hKey&)


Global Const HKEY_CLASSES_ROOT = 1

Global Const MAX_PATH = 128

Global Const REG_SZ = 1


' return codes from Registration functions

Global Const ERROR_SUCCESS = 0&

Global Const ERROR_BADDB = 1&

Global Const ERROR_BADKEY = 2&

Global Const ERROR_CANTOPEN = 3&

Global Const ERROR_CANTREAD = 4&

Global Const ERROR_CANTWRITE = 5&

Global Const ERROR_OUTOFMEMORY = 6&

Global Const ERROR_INVALID_PARAMETER = 7&

Global Const ERROR_ACCESS_DENIED = 8&


Function AddKey (sKeyName As String, sKeyValue As String) As Long

    ret& = RegCreateKey(HKEY_CLASSES_ROOT, sKeyName, lphKey&)

    ret& = RegSetValue(lphKey&, "", REG_SZ, sKeyValue, 0&)

    AddKey = ret&
```

```
End Function

Sub EnumRoot ()
    Dim sSubKey As String * MAX_PATH
    Dim sValue As String * MAX_PATH
    Dim sKeyVal As String
    BuffLen& = CLng(MAX_PATH)

    For i& = 0 To 1000
        sSubKey = String(MAX_PATH, " ")
        sValue = String(MAX_PATH, " ")
        BuffLen& = CLng(MAX_PATH)
        ret& = RegEnumKey(HKEY_CLASSES_ROOT, i&, sSubKey, BuffLen&)
        If ret& <> 0 Then Exit For
        sKeyVal = Left(sSubKey, Len(Trim$(sSubKey)) - 1)
        ret& = RegQueryValue(HKEY_CLASSES_ROOT, sKeyVal, sValue, BuffLen&)
        Debug.Print Left$(sValue, BuffLen&)
    Next i&

End Sub

Sub GetCurVer ()
    Dim sSubKey As String * MAX_PATH
    Dim sValue As String * MAX_PATH
    Dim sKeyVal As String
    BuffLen& = CLng(MAX_PATH)

    For i& = 0 To 1000
        sSubKey = String(MAX_PATH, " ")
        sValue = String(MAX_PATH, " ")
        BuffLen& = CLng(MAX_PATH)
        ret& = RegEnumKey(HKEY_CLASSES_ROOT, i&, sSubKey, BuffLen&)
        If ret& <> 0 Then Exit For
```

```
        sKeyVal = Left(sSubKey, Len(Trim$(sSubKey)) - 1)
        ret& = RegOpenKey(HKEY_CLASSES_ROOT, sKeyVal, hkSFE&)
        ret& = RegQueryValue(hkSFE&, "CurVer", sValue, BuffLen&)
        If ret& = 0 Then
            Debug.Print "CurrentVersion = " & Left$(sValue, BuffLen&)
        End If
        ret& = RegCloseKey(hkSFE&)
    Next i&
End Sub

Function GetKey (RegPath As String)
    Dim RegValue As String * 128
    success& = RegQueryValue(HKEY_CLASSES_ROOT, RegPath + Chr$(0), RegValue,
128)
    GetKey = Left(RegValue, InStr(1, RegValue, Chr$(0)) - 1)
End Function

Sub GetServer ()
    Dim sSubKey As String * MAX_PATH
    Dim sValue As String * MAX_PATH
    Dim sKeyVal As String
    BuffLen& = CLng(MAX_PATH)

    For i& = 0 To 10
        sSubKey = String(MAX_PATH, " ")
        sValue = String(MAX_PATH, " ")
        BuffLen& = CLng(MAX_PATH)
        ret& = RegEnumKey(HKEY_CLASSES_ROOT, i&, sSubKey, BuffLen&)
        If ret& <> 0 Then Exit For
        sKeyVal = Left(sSubKey, Len(Trim$(sSubKey)) - 1)
        ret& = RegOpenKey(HKEY_CLASSES_ROOT, sKeyVal &
"\protocol\StdFileEditing", hkSFE&)
        ret& = RegQueryValue(hkSFE&, "server", sValue, BuffLen&)
```

```
        If ret& = 0 Then
            Debug.Print "KeyVal=" & sKeyVal
            Debug.Print "  Key=" & hkSFE&, " Server= " & Left$(sValue,
BuffLen&)
        End If
        ret& = RegCloseKey(hkSFE&)
    Next i&

End Sub
```

SendMail

```
'Declares for SendMail
Type MAPIMessage
    Reserved As Long
    Subject As String
    NoteText As String
    MessageType As String
    DateReceived As String
    ConversationID As String
    Flags As Long
    RecipCount As Long
    FileCount As Long
End Type

Type MapiRecip
    Reserved As Long
    RecipClass As Long
    Name As String
    Address As String
    EIDSize As Long
    EntryID As String
End Type

Type MapiFile
    Reserved As Long
    Flags As Long
    Position As Long
    PathName As String
    FileName As String
    FileType As String
End Type
```

```
Global M As MAPIMessage

Global Mr(0 To 1) As MapiRecip

Global Mf(0 To 1) As MapiFile


'***************************

'   FUNCTION Declarations

'***************************


Declare Function MAPILogon Lib "MAPI.DLL" (ByVal UIParam&, ByVal User$, ByVal
Password$, ByVal Flags&, ByVal Reserved&, Session&) As Long

Declare Function MAPILogoff Lib "MAPI.DLL" (ByVal Session&, ByVal UIParam&,
ByVal Flags&, ByVal Reserved&) As Long

Declare Function MAPISendMail Lib "MAPI.DLL" Alias "BMAPISendMail" (ByVal
Session&, ByVal UIParam&, Message As MAPIMessage, Recipient As MapiRecip,
File As MapiFile, ByVal Flags&, ByVal Reserved&) As Long

Declare Function GetModuleHandle Lib "Kernel" (ByVal FileName$) As Integer

Declare Function GetModuleUsage Lib "Kernel" (ByVal hModule%) As Integer


'***************************

'   CONSTANT Declarations

'***************************


Global Const SUCCESS_SUCCESS = 0

Global Const MAPI_USER_ABORT = 1

Global Const MAPI_E_FAILURE = 2

Global Const MAPI_E_LOGIN_FAILURE = 3

Global Const MAPI_E_DISK_FULL = 4

Global Const MAPI_E_INSUFFICIENT_MEMORY = 5

Global Const MAPI_E_BLK_TOO_SMALL = 6

Global Const MAPI_E_TOO_MANY_SESSIONS = 8

Global Const MAPI_E_TOO_MANY_FILES = 9

Global Const MAPI_E_TOO_MANY_RECIPIENTS = 10

Global Const MAPI_E_ATTACHMENT_NOT_FOUND = 11

Global Const MAPI_E_ATTACHMENT_OPEN_FAILURE = 12
```

```
Global Const MAPI_E_ATTACHMENT_WRITE_FAILURE = 13
Global Const MAPI_E_UNKNOWN_RECIPIENT = 14
Global Const MAPI_E_BAD_RECIPTYPE = 15
Global Const MAPI_E_NO_MESSAGES = 16
Global Const MAPI_E_INVALID_MESSAGE = 17
Global Const MAPI_E_TEXT_TOO_LARGE = 18
Global Const MAPI_E_INVALID_SESSION = 19
Global Const MAPI_E_TYPE_NOT_SUPPORTED = 20
Global Const MAPI_E_AMBIGUOUS_RECIPIENT = 21

Global Const MAPI_ORIG = 0
Global Const MAPI_TO = 1
Global Const MAPI_CC = 2
Global Const MAPI_BCC = 3


'************************
'   FLAG Declarations
'************************

Global Const MAPI_LOGON_UI = &H1
Global Const MAPI_NEW_SESSION = &H2
Global Const MAPI_DIALOG = &H8
Global Const MAPI_UNREAD_ONLY = &H20
Global Const MAPI_ENVELOPE_ONLY = &H40
Global Const MAPI_PEEK = &H80
Global Const MAPI_GUARANTEE_FIFO = &H100
Global Const MAPI_BODY_AS_FILE = &H200
Global Const MAPI_AB_NOMODIFY = &H400
Global Const MAPI_SUPPRESS_ATTACH = &H800
Global Const MAPI_FORCE_DOWNLOAD = &H1000

Global Const MAPI_OLE = &H1
```

```
Global Const MAPI_OLE_STATIC = &H2


Function sendmail (EMailAlias As String, SubjectLine As String, MessageText
As String) As Long


    '****************************************
    ' Set up the structures
    '****************************************


    Mr(0).Name = EMailAlias
    Mr(0).RecipClass = 1   'Always a 1
    M.RecipCount = 1   'How many email names to send to (You can change it to
1 to only send mail to one person
    M.FileCount = 0
    M.Reserved = 0   'Always 0
    M.Subject = SubjectLine
    M.NoteText = MessageText
    M.MessageType = ""  'These next three lines are default
    M.DateReceived = ""
    M.Flags = 0


    rc& = MAPISendMail(0, 0, M, Mr(0), Mf(0), 0, 0)


    If rc& = 0 Then
        ' Do nothing - Successful! (Already logged on)
    ElseIf rc& = 3 Then


    '******************
    ' Log the user on.
    '******************


    'This opens and closes only 1 mail 'session'
    flag& = MAPI_LOGON_UI
```

```
    rc& = MAPILogon(0, "", "", flag&, 0&, MapiSession&)

    'Once they log on, send it again
    rc& = MAPISendMail(0, 0, M, Mr(0), Mf(0), 0, 0)

    'If they cancel out of logon then display a message
    If rc& <> 0 Then MsgBox "Could not send mail - Canceled", , "ERROR"

    'Log off the session
    rc& = MAPILogoff(MapiSession&, 0, 0, 0)
    MapiSession& = 0&

    Else
        MsgBox "Please make sure you are sending this from your mail
machine", , "ERROR SENDING MESSAGE"
    End If

sendmail = rc&

End Function
```

CheckPeriod

```
Sub CheckPeriod (t As Control)
    Dim i As Integer

    i = InStr(1, t.Text, ".")   ' Look for a period
    If i > 0 And InStr(i + 1, t.Text, ".") > 0 Then
        t.SelStart = t.SelStart - 1
        t.SelLength = 1         ' Select new period
        t.SelText = ""          ' Remove new period
    End If
End Sub
```

Default to Integer

```
DefInt A-Z
```

HighWord

```
Function HIWORD%(LongVal&)
    HIWORD% = LongVal& \ 65536 ' (note: '\', not '/')
End Function
```

LoWord

```
Function LoWord%(LongVal&)
    LOWORD% = LongVal& AND 65535
End Function
```

NumbersOnly

```
Sub NumbersOnly (t As Control, KeyAscii As Integer)
    '
    ' This subroutine discards any characters that can't be in
    ' a number.  Here are the allowed characters:
    '
    '     0..9     All digits are allowed
    '     -        A minus, only if it's the first character
    '     .        Periods are allowed (they're checked in KeyUp)
    '
    If KeyAscii < Asc(" ") Then     ' Is this Control char?
        Exit Sub                    ' Yes, let it pass
    End If

    CheckPeriod t                   ' Remove excess periods

    If KeyAscii >= Asc("0") And KeyAscii <= Asc("9") Then
        ' keep digit
    ElseIf KeyAscii = Asc(".") Then
        ' keep .
    ElseIf KeyAscii = Asc("-") And t.SelStart = 0 Then
        ' Keep - only if first char
    Else
        KeyAscii = 0                ' Discard all other chars
    End If

    '
    ' This code keeps you from typing any characters in front of
    ' a minus sign.
    '
    If Mid$(t.Text, t.SelStart + t.SelLength + 1, 1) = "-" Then
        KeyAscii = 0                ' Discard chars before -
```

```
    End If
End Sub
```

FixAPI

```
Function FixAPIString$ (ByVal test$)
    FixAPIString$ = Trim(Left$(test$, InStr(test$, Chr$(0)) - 1))
End Function
```

GetWorkStationInfo

```
'Declares for GetWorkStationInfo

Declare Function NetWkstaGetInfo% Lib "NetAPI.DLL" (ByVal lServer&, ByVal
sLevel%, ByVal pbBuffer&, ByVal cbBuffer%, pcbTotalAvail%)

Declare Function GlobalAlloc% Lib "Kernel" (ByVal fFlags%, ByVal nSize&)

Declare Function GlobalLock& Lib "Kernel" (ByVal hMem%)

Declare Function GlobalUnlock% Lib "Kernel" (ByVal hMem%)

Declare Function GlobalFree% Lib "Kernel" (ByVal hMem%)

Declare Sub lstrcpy Lib "Kernel" (ByVal dest As Any, ByVal src As Any)

Declare Sub hmemcpy Lib "Kernel" (ByVal dest As Any, ByVal src As Any, ByVal
size As Long)


Function GetBinInt (sobj As String, off As Integer)

    If (Asc(Mid$(sobj, off + 1, 1)) < 128) Then

       GetBinInt = Asc(Mid$(sobj, off, 1)) + Asc(Mid$(sobj, off + 1, 1)) *
256

     Else

       GetBinInt = ((&HFF - Asc(Mid$(sobj, off + 1, 1))) * 256) - Asc(Mid$
(sobj, off, 1))

     End If

End Function


Function GetWorkStationInfo (sComputer As String, sUserName As String,
sWorkgroup As String, sLogonDomain As String)

     Dim nRetSize As Integer, nStruct, ret As Integer, dummy As Integer

     Dim hMem As Integer, lpMem As Long, sTemp As String


     ' Get the size of the return structure.

     ret = NetWkstaGetInfo(0&, 10, 0&, 0, nRetSize)

     If (ret <> 0) And (ret <> 2123) Then

        GetWorkstationInfo = False

        Exit Function

     End If
```

```
    ' Allocate memory for the structure.
    hMem = GlobalAlloc(0, CLng(nRetSize))
    nStruct = nRetSize
    If (hMem <> 0) Then
       lpMem = GlobalLock(hMem)

       ' Read workstation information into structure.
       ret = NetWkstaGetInfo(0&, 10, lpMem, nRetSize, nRetSize)
       If (ret = 0) Then
          sTemp = Space(100)
          hmemcpy sTemp, lpMem, nStruct

       'Retrieve the computer name.
          sComputer = Space(100)
          lpMem = CLng(GetBinInt(sTemp, 1)) + (CLng(GetBinInt(sTemp, 3)) *
CLng(&H10000))
          lstrcpy sComputer, lpMem
          sComputer = Mid(sComputer, 1, InStr(sComputer, Chr(0)) - 1)

          ' Retrieve the user name.
          sUserName = Space(100)
          lpMem = CLng(GetBinInt(sTemp, 5)) + (CLng(GetBinInt(sTemp, 7)) *
&H10000)
          lstrcpy sUserName, lpMem
          sUserName = Mid(sUserName, 1, InStr(sUserName, Chr(0)) - 1)

          ' Retrieve the workgroup name.
          sWorkgroup = Space(100)
          lpMem = CLng(GetBinInt(sTemp, 9)) + (CLng(GetBinInt(sTemp, 11)) *
&H10000)
          lstrcpy sWorkgroup, lpMem
          sWorkgroup = Mid(sWorkgroup, 1, InStr(sWorkgroup, Chr(0)) - 1)

          ' Retrieve the logon domain name.
```

```
            sLogonDomain = Space(100)
            lpMem = CLng(GetBinInt(sTemp, 15)) + (CLng(GetBinInt(sTemp, 17)) *
&H10000)
            lstrcpy sLogonDomain, lpMem
            sLogonDomain = Mid(sLogonDomain, 1, InStr(sLogonDomain, Chr(0)) -
1)
         End If
         ' Free the memory allocated.
         dummy = GlobalUnlock(hMem)
         dummy = GlobalFree(hMem)
      Else
         ret = -1
      End If
      GetWorkstationInfo = IIf(ret = 0, True, False)
End Function
```

Net Connections

```
'Declares for Net Connections

Declare Function WNetAddConnection% Lib "User" (ByVal lpszNetPath$, ByVal
lpszPassword$, ByVal lpszDrive$)

Declare Function WNetCancelConnection% Lib "User" (ByVal lpszName$, ByVal
fForce%)


Function Connect (Path$, Drive$) As Integer

    Connect = WNetAddConnection%(Path$, "", Drive$)

End Function


Function Disconnect (Drive$) As Integer

    Disconnect = WNetCancelConnection%(Drive$, 0)

End Function
```

MakeBeep

```
Sub MakeBeep (Reps%)
    For X=1 to Reps%
        Beep
    Next
End Sub
```

PlayWaveFile

```
'Declares for PlayWaveFile

Declare Sub sndPlaySound Lib "MMSYSTEM" (ByVal WavFile As String, ByVal
wFlags As Integer)


Sub PlayWaveFile (varWave)

    sndPlaySound varWave, 1

End Sub
```

Elapse

```
Function Elapse ()
Static c As Integer, Start As Double, Done As Double
c = c + 1
If c = 1 Then
    Start = Timer
Else
    Done = Timer - Start
    MsgBox "Time Elapsed: " & Done
End If
End Function
```

WaitSecs

```
Sub WaitSecs (secs)
    Dim sTart!, Temp%
    start! = Timer
    While Timer < start! + secs +1
         Temp% = DoEvents()
    Wend
End Sub
```

FindParent

```
'Declares for Find Parent
Declare Function GetParent Lib "User" (ByVal hWnd As Integer) As Integer


Function FindProgentor (WinHand As Integer) As Integer
     Parent% = GetParent(WinHand%)
     OldParent%=Parent%
     'Get the parent of the parent if any
     Do While Parent%
          OldParent% = Parent%
          Parent% = GetParent%(OldParent%)
         ' Debug.Print Parent%
     Loop
Parent%=OldParent%
FindProgenitor = Parent%
End Function
```

FindAndRestorePrevInstance

```
Sub FindAndRestorePrevInstance (Cap$)
    Dim X%
    If App.PrevInstance Then
       AppActivate Cap$
       SendKeys ("% R")
       End
    End If
End Sub
```

Loaded

```
'Declares for Loaded

Declare Function FindWindow Lib "user" (ByVal CName As Any, ByVal Caption As
Any)


Function Loaded (ClassName$)

    Loaded = FindWindow(ClassName$, 0&)

End Function
```

RestoreApp

```
'Declares for RestoreApp
Declare Function IsIconic Lib "user" (ByVal hWnd As Any)

Sub RestoreApp (wHandle)
    WM_SYSCOMMAND = &H112
    SC_RESTORE = &HF120
    If IsIconic(Instance) Then
        T = PostMessage(Instance, WM_SYSCOMMAND, SC_RESTORE, 0)
        WaitSecs 1
    End If
End Sub
```

SearchWindowList

```
'Declares for SearchWindowLIst

Declare Function GetWindow% Lib "USER" (ByVal hWnd%, ByVal wCmd%)

Global Const GW_HWNDFIRST = 0

Global Const GW_HWNDNEXT = 2

Declare Function GetWindowText Lib "User" (ByVal hWnd As Integer, ByVal
lpString As String, ByVal aint As Integer) As Integer


Function SearchWindowList% (Cap$)

    'returns handle of first window that matches partial

    'caption passed to function

    SearchWindowList% = 0

    Dim w%, Y%, winCap As String * 255

    w% = GetWindow%(MAKerMain.hWnd, GW_HWNDFIRST)

    Do While w% <> 0

        Y% = GetWindowText(w%, winCap, 254)

            If Left$(winCap, Len(Cap$)) = Cap$ Then

                SearchWindowList% = w%

                Exit Do

            End If

            w% = GetWindow%(w%, GW_HWNDNEXT)

    Loop

End Function
```

TrackPopupMenu

```
'TrackPopupMenu declares
Declare Function TrackPopupMenu% Lib "user" (ByVal hMenu%, ByVal
wFlags%, ByVal X%, ByVal Y%, ByVal r2%, ByVal hWnd%, ByVal r1&)
Declare Function GetMenu% Lib "user" (ByVal hWnd%)
Declare Function GetSubMenu% Lib "user" (ByVal hMenu%, ByVal nPos%)


Sub TrackPopUp (Menu As Integer, X As Single, Y As Single, F as Form,
MenuForm As Form)
          Const PIXEL = 3
          Const TWIP = 1
          F.ScaleMode = PIXEL
          InPixels = F.ScaleWidth
          F.ScaleMode = TWIP
          ix = (X + F.Left) \ (F.ScaleWidth \ InPixels)
          iy = (Y + (F.Top + (F.Height - F.ScaleHeight - (F.Width -
F.ScaleWidth)))) \ (F.ScaleWidth \ InPixels)
          hMenu% = GetMenu(MenuForm.hWnd)
          hSubMenu% = GetSubMenu(hMenu%, Menu)
          '2 tells it to use right mouse button, 1 the left button
          r = TrackPopupMenu(hSubMenu%, 2, ix, iy, 0, MenuForm.hWnd, 0)
End Sub
```

GetWindowsVersion

GetWindowsVersion
How VB Can Get Windows Status Information via API Calls [vbwin]
ID: Q84556 CREATED: 14-MAY-1992 MODIFIED: 21-JUN-1995

SUMMARY

The Visual Basic for Windows program example below demonstrates how you can obtain system status information similar to the information displayed in the Windows Program Manager About box. The example program displays the following information using the Windows API function(s) indicated:

- The Windows version number with GetVersion
- The kind of CPU (80286, 80386, or 80486) and whether a math coprocessor is present with GetWinFlags
- Whether Windows is running in enhanced mode or standard mode with

  GetWinFlags
- The amount of free memory with GetFreeSpace and GlobalCompact
- The percentage of free system resources with SystemHeapInfo

NOTE: The API function SystemHeapInfo is new to Windows version 3.1 and is not available in Windows, version 3.0. All other API functions listed above are available in both Windows versions 3.0 or 3.1.

MORE INFORMATION

Steps to Create Example Program

1. Run Visual Basic for Windows, or if Visual Basic for Windows is already running, choose New Project from the File menu (press ALT, F, N). Form1 will be created by default.

2. From the File menu, choose Add Module (press ALT, F, M). Module 1 is created by default (In Visual Basic version 1.0 for Windows, this step is unnecessary).

3. Enter the following code into the general declarations section of a code module (In Visual Basic version 1.0 for Windows, place the following in the Global module):

```
' Constants for GetWinFlags.
Global Const WF_CPU286 = &H2
Global Const WF_CPU386 = &H4
Global Const WF_CPU486 = &H8
```

```
Global Const WF_80x87 = &H400
Global Const WF_STANDARD = &H10
Global Const WF_ENHANCED = &H20
Global Const WF_WINNT = &H4000

' Type for SystemHeapInfo.
Type SYSHEAPINFO
   dwSize As Long
   wUserFreePercent As Integer
   wGDIFreePercent As Integer
   hUserSegment As Integer

   hGDISegment As Integer
End Type

Declare Function GetVersion Lib "Kernel" () As Integer
Declare Function GetWinFlags Lib "Kernel" () As Long
'Enter each of the following Declare statements as one, single line:
Declare Function GetFreeSpace Lib "Kernel" (ByVal wFlags As Integer)
   As Long
Declare Function GlobalCompact Lib "Kernel" (ByVal dwMinFree As Long)
   As Long
Declare Function SystemHeapInfo Lib "toolhelp.dll" (shi As
   SYSHEAPINFO) As Integer
```

4. Enter the following code into the Form_Load procedure of Form1:

```
Sub Form_Load ()
   Dim msg As String         ' Status information.
   Dim nl As String          ' New-line.
   nl = Chr$(13) + Chr$(10)  ' New-line.

   Show
   MousePointer = 11   ' Hourglass.
   ver% = GetVersion()
   status& = GetWinFlags()

   ' Get operating system and version.
   If status& And WF_WINNT Then
      msg = msg + "Microsoft Windows NT "

   Else
      msg = msg + "Microsoft Windows "
   End If
   ver_major$ = Format$(ver% And &HFF)
   ver_minor$ = Format$(ver% \ &H100, "00")
   msg = msg + ver_major$ + "." + ver_minor$ + nl

   ' Get CPU kind and operating mode.
   msg = msg + "CPU: "
   If status& And WF_CPU286 Then msg = msg + "80286"
   If status& And WF_CPU386 Then msg = msg + "80386"
   If status& And WF_CPU486 Then msg = msg + "80486"
   If status& And WF_80x87 Then msg = msg + " with 80x87"

   msg = msg + nl
   msg = msg + "Mode: "
```

```
    If status& And WF_STANDARD Then msg = msg + "Standard" + nl
    If status& And WF_ENHANCED Then msg = msg + "Enhanced" + nl

    ' Get free memory.
    memory& = GetFreeSpace(0)
    msg = msg + "Memory free: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl
    memory& = GlobalCompact(&HFFFFFFFF)
    msg = msg + "Largest free block: "
    msg = msg + Format$(memory& \ 1024, "###,###,###") + "K" + nl

    ' Get free system resources.
    ' The API SystemHeapInfo became available in Windows version 3.1.
    msg = msg + "System resources: "
    If ver% >= &H310 Then
       Dim shi As SYSHEAPINFO
       shi.dwSize = Len(shi)
       If SystemHeapInfo(shi) Then
          If shi.wUserFreePercent < shi.wGDIFreePercent Then
             msg = msg + Format$(shi.wUserFreePercent) + "%"
          Else
             msg = msg + Format$(shi.wGDIFreePercent) + "%"

          End If
       End If
    Else
       msg = msg + "n/a"
    End If

    MsgBox msg, 0, "About " + Caption
    MousePointer = 0
End Sub
```

5. Press the F5 key to run the program.

PutIni

```
'Declares for PutIni
Declare Function WritePrivateProfileString Lib "Kernel" (ByVal
lpApplicationName As String, ByVal lpKeyName As String, ByVal lpSet$,
ByVal lpFileName$) As Integer

Function PutIni (FileName As String, Section As String, Keyword As
String, Value As String)
    x = WritePrivateProfileString(Section, Keyword, Value, FileName)
End Function
```

Jeff Williams (Microsoft Premier Applications)

I wrote a lot of this code myself but I did get some from these sources also:

Visual Baisc Utilities by Paul Bonner
Ziff-Davis Press ($29.95)

Visual Basic Programmers Guide to the Windows API
by Daniel Appleman
PC Magazine ($34.95)

Visual Basic Power ToolKit by Mansfield and Petroutsos
Ventana Press ($39.95)

Teach Yourself Visual Basic by John Socha and Devra Hall
MIS Press ($24.95)

Master Visual Basic 3 by Gurewich and Gurewich
SAMS Publishing ($45.00)

The MAPI developers kit from Microsoft
Sales: 1-800-426-9400

If you have any questions or need any info on updates you can reach me at **`jeffwill@microsoft.com`** on the Internet.