Take the Long File Home

Click & Retrieve Source CODE!

Give your apps access to long file names and learn about OLE strings, thunking, and common dialogs in the process.

by Jonathan Zuck

Even if you don't remember the rock group Supertramp, you most likely remember last year's column about using the Windows APIs to get at the File Open and File Save As common dialogs. The common dialog VBX (and now OCX) is just a thin wrapper around the common dialog API functions. In the case of the file dialogs, the functions are GetOpenFileName and GetSaveFileName. These functions live on in the Win32 API, but now they provide an "Explorer" style dialog and access to long file names.

If you do recall the piece [Windows Programming, "Uncommon Dialogs," *VBPJ* February 1995], you might remember that the "pure" VB approach, with a couple of limitations due to the lack of callbacks, allowed you to use the common dialogs without a custom control, thereby eliminating the overhead and installation of one more custom control with your applications. This month, I'd like to revisit the common dialog functions with three goals in mind: demonstrate their use in Win32, provide access to the 32-bit versions of the dialogs and long file names from VB3 and 16-bit VB4, and wrap it all up with a class similar to the common dialog to facilitate migration from 16- to 32-bit.

The code in that previous article was a little messy, because the OPENFILENAME structure requires pointers to strings. As you will see, such machinations are no longer necessary because the internal storage of VB strings has changed significantly and has become more compatible with external processes.

The OPENFILENAME structure is defined this way in C:

```
typedef struct tagOPENFILENAME { /* ofn */
   DWORD       lStructSize;
   HWND        hwndOwner;
   HINSTANCE   hInstance;
   LPCSTR      lpstrFilter;
   LPSTR       lpstrCustomFilter;
   DWORD       nMaxCustFilter;
   DWORD       nFilterIndex;
   LPSTR       lpstrFile;
   DWORD       nMaxFile;
   LPSTR       lpstrFileTitle;
   DWORD       nMaxFileTitle;
   LPCSTR      lpstrInitialDir;
   LPCSTR      lpstrTitle;
   DWORD       Flags;
   UINT        nFileOffset;
   UINT        nFileExtension;
   LPCSTR      lpstrDefExt;
   LPARAM      lCustData;
   UINT        (CALLBACK* lpfnHook) (HWND, UINT, WPARAM,
               LPARAM);

   LPCSTR      lpTemplateName;
} OPENFILENAME;
```

An LPCSTR is a long pointer to a null-terminated string. In VB3, strings aren't naturally null terminated, and you also had no easy way to store the address of a string into a structure element. Therefore, you had to declare these elements as Long integers and use the VarPtr function (exported from the VB runtime DLL) to determine the address of the string. Luckily, by passing a string by value to the VarPtr function, you accomplished two things. First, passing ByVal dereferences the string and passes a long pointer to the string data. Second, it appends a null character. So you were able to populate these long-integer elements with an address to a null-terminated string. Not the most fun you've ever had, but doable.

Well, with VB4 the situation is greatly improved. Instead of being proprietary, VB strings are managed internally as OLE strings (BSTRING). Just like the HLSTR of yesteryear, the BSTRING has a set of functions you should use to manipulate it from within a DLL. The difference is that these are published interfaces, using exported functions. Much nicer. In addition, the BSTRING has another interesting property. It is stored internally a little like an LPCSTR. The BSTRING is a pointer to the actual string data and the string is maintained internally with a null on the end. Unlike the HLSTR, which was a pointer to a pointer to a string length that was followed by the string data, the BSTR is a pointer to the actual string data, which is proceeded—at least for now—by the string length (see Figure 1).

Therefore, joy of joys, you can use a BSTR as an LPCSTR both as a parameter to a DLL but, more importantly, as a member of a structure. The OPENFILENAME type definition in VB4 looks this:

```
Private Type OPENFILENAME
```

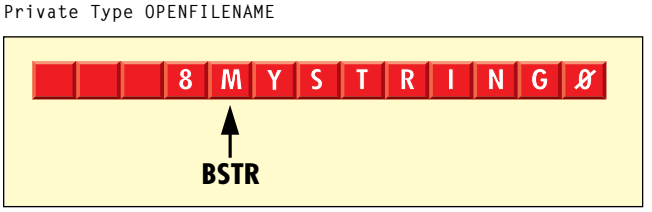

FIGURE 1 ***Let Me Point It Out.*** *The BSTR is a pointer to the string data, which is proceeded by the string length. The whole thing is null terminated for easy use as an LPSTR.*

Jonathan Zuck is vice president of client/server technology for Advanced Paradigms Inc. in Alexandria, Virginia. He is a regular contributor to Visual Basic Programmer's Journal *and a coauthor of the first* Visual Basic How-To*, published by Waite Group Press. Reach Jonathan on CompuServe at 76702,1605.*

VB4

```
VERSION 1.0 CLASS
BEGIN
  MultiUse = -1  'True
END
Attribute VB_Name = "CommonDialog"
Attribute VB_Creatable = True
Attribute VB_Exposed = True
Option Explicit

#If Win32 Then
  Private Type OPENFILENAME
        lStructSize As Long
        hwndOwner As Long
        hInstance As Long
        lpstrFilter As String
        lpstrCustomFilter As String
        nMaxCustFilter As Long
        nFilterIndex As Long
        lpstrFile As String
        nMaxFile As Long
        lpstrFileTitle As String
        nMaxFileTitle As Long
        lpstrInitialDir As String
        lpstrTitle As String
        Flags As Long
        nFileOffset As Integer
        nFileExtension As Integer
        lpstrDefExt As String
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
  End Type

  Private Declare Function GetOpenFileName Lib _
     "comdlg32.dll" Alias "GetOpenFileNameA" _
     (pOPENFILENAME As OPENFILENAME) As Long
  Private Declare Function GetSaveFileName Lib _
     "comdlg32.dll" Alias "GetSaveFileNameA" _
     (pOPENFILENAME As OPENFILENAME) As Long
  Private Declare Function GetShortPathName Lib _
     "kernel32" Alias "GetShortPathNameA" (ByVal _
     lpszLongPath As String, ByVal lpszShortPath As _
     String, ByVal cchBuffer As Long) As Long
  Private Declare Function GetActiveWindow Lib _
     "user32" () As Long
#Else
  Private Type OPENFILENAME
        lStructSize As Long
        hwndOwner As Integer
        hInstance As Integer
        lpstrFilter As String
        lpstrCustomFilter As String
        nMaxCustFilter As Long
        nFilterIndex As Long
        lpstrFile As String
        nMaxFile As Long
        lpstrFileTitle As String
        nMaxFileTitle As Long
        lpstrInitialDir As String
        lpstrTitle As String
        Flags As Long
        nFileOffset As Integer
        nFileExtension As Integer
        lpstrDefExt As String
        lCustData As Long
        lpfnHook As Long
        lpTemplateName As String
  End Type

  Private Declare Function GetOpenFileName Lib _
     "commdlg.dll" (pOPENFILENAME As OPENFILENAME) _
     As Long
  Private Declare Function GetSaveFileName Lib _
     "commdlg.dll" (pOPENFILENAME As OPENFILENAME) _
     As Long
  Private Declare Function GetActiveWindow Lib "user" _
     () As Integer
#End If

'here are some direct properties
Public DefaultExt As String
Public DialogTitle As String
Public Filter As String
Public FilterIndex As String
Public Flags As Integer
Public InitDir As String

'member variables
Dim mCMDLG As Object
Dim mFileName As String
Dim mFileTitle As String
Dim mhOwner As Long

Dim NullChar As String

Public Property Let Action(Index As Integer)
  Dim OFN As OPENFILENAME, sFile As String, lResult _
     As Long, iDelim As Integer
  Dim zTemp As String, Temp As Variant
  Dim i As Integer

  If Index > 2 Then Exit Property  'get out if invalid

  OFN.lStructSize = Len(OFN)
  If mhOwner = 0 Then mhOwner = GetActiveWindow()
  OFN.hwndOwner = mhOwner
  OFN.Flags = Flags

  OFN.lpstrDefExt = DefaultExt

  'set the initial directory, otherwise uses current
  Temp = InitDir
  OFN.lpstrInitialDir = Temp

  'retrieve the default file name\
  'first check for wild cards
  Temp = mFileName

  #If Win32 Then
     If (InStr(Temp, "*") = 0) And InStr(Temp, "?") _
        = 0 Then
        'try to convert it to a long file name
        zTemp = Dir(OFN.lpstrInitialDir & "\" & Temp)
        If Len(zTemp) Then 'we found a match
           Temp = zTemp
        End If
     End If
  #End If

  OFN.lpstrFile = Temp & String$(255 - Len(Temp), 0)
  OFN.nMaxFile = 255

  OFN.lpstrFileTitle = String$(255, 0)
  OFN.nMaxFileTitle = 255

  'file type filter
  'we need to replace pipes with nulls
  zTemp = Filter
  For i = 1 To Len(zTemp)
     If Mid(zTemp, i, 1) = "|" Then Mid(zTemp, i, 1) _
        = NullChar
  Next
  zTemp = zTemp & String$(2, 0)
  OFN.lpstrFilter = zTemp
  OFN.nFilterIndex = FilterIndex

  OFN.lpstrTitle = DialogTitle

  If Index = 1 Then 'they want File Open dialog
```

CONTINUED ON NEXT PAGE.

LISTING 1 ***Common but Classy.*** *You can use this common dialog class, CMDLGX.CLS, as an OLE-server replacement for the common dialog control in VB3 and VB4.*


```
    lStructSize As Long
    hwndOwner As Long
    hInstance As Long
    lpstrFilter As String
    lpstrCustomFilter As String
    nMaxCustFilter As Long
    nFilterIndex As Long
    lpstrFile As String
    nMaxFile As Long
    lpstrFileTitle As String
    nMaxFileTitle As Long
    lpstrInitialDir As String
    lpstrTitle As String
    Flags As Long
    nFileOffset As Integer
    nFileExtension As Integer
    lpstrDefExt As String
    lCustData As Long
    lpfnHook As Long
    lpTemplateName As String
End Type
```

Be advised, of course, that as before, you need to pad these strings for incoming data because a DLL function expecting an LPSTR expects space to be preallocated and will not modify the length of the string.

DIALOGS WITH CLASS

The obvious thing to do, when implementing the common dialog functions in VB, is to create a class with an object model that's compatible with the common dialog control so that you can easily use it as a replacement for the control (see Listing 1).

The operation of this CommonDialog class is pretty straightforward, but it demonstrates a number of concepts. One of the more interesting things about it is that a 16-bit application running under a 32-bit OS can use it. The thunking between 16- and 32-bit is a byproduct of OLE, but I've added some goodies to facilitate the use of long file names. Therefore, when you implement this class as an OLE Automation server under both 16- and 32-bit, not only can 16- or 32-bit applications use it, but it will also show the "best" dialog for the OS it's running on.

The code for the class begins by conditionally defining the OPENFILENAME structure and the necessary Windows API declarations. No surprises here. Stuff defined as unsigned integer (UINT) in the C structure are short integers in 16-bit and long integers in 32-bit. Next you have some simple properties, which you create using the Public keyword:

```
Public DefaultExt As String
Public DialogTitle As String
Public Filter As String
Public FilterIndex As String
Public Flags As Integer
Public InitDir As String
```

The variables that begin with "m" are member variables (known as fields in Delphi). You need to keep these properties hidden and provide access to them only through property procedures. You can set and read most of these properties with no problem:

```
Dim mCMDLG As Object
Dim mFileName As String
Dim mFileTitle As String
Dim mhOwner As Long
```

Just for the fun of it, I designed the class to be used in conjunction with the control. For those who want to do most of your work in the property editor rather than in code, this class allows you, to a limited extent, to accomplish that. You can use the Set statement to assign the CMDLGControl property, one of

CONTINUED FROM PREVIOUS PAGE.

```
    lResult = GetOpenFileName(OFN)
  Else           'Save As... dialog
    lResult = GetSaveFileName(OFN)
  End If

  If lResult <> 0 Then
    iDelim = InStr(OFN.lpstrFileTitle, NullChar)
    If iDelim > 0 Then
      mFileTitle = Left$(OFN.lpstrFileTitle, iDelim - 1)
    End If
    iDelim = InStr(OFN.lpstrFile, NullChar)
    If iDelim > 0 Then
      mFileName = Left$(OFN.lpstrFile, iDelim - 1)
    End If
  End If
End Property

Public Property Set CMDLGControl(C As Object)
  Set mCMDLG = C
  FileName = mCMDLG.FileName
  DefaultExt = mCMDLG.DefaultExt
  Filter = mCMDLG.Filter
  FilterIndex = mCMDLG.FilterIndex
  Flags = mCMDLG.Flags
  InitDir = mCMDLG.InitDir
  mhOwner = mCMDLG.Parent.hWnd
End Property

Public Property Let FileName(S As String)
  mFileName = S
End Property

Public Property Get FileName() As String
  FileName = mFileName
End Property

Public Property Get ShortFileTitle() As String
  ShortFileTitle = Long2Short(mFileTitle)
End Property

Public Property Get FileTitle() As String
  FileTitle = mFileTitle
End Property

Private Function Long2Short(ByVal S As String) As
String
  Dim Buff As String
  Dim r As Integer

  #If Win32 Then
    If Dir(S) = "" Then
      Open S For Output As #1
      Close
    End If
    Buff = Space(256)
    r = GetShortPathName(S, Buff, 256)
    Long2Short = Left(Buff, r)
  #Else
    Long2Short = S
  #End If
End Function

Public Property Get ShortFileName() As String
  ShortFileName = Long2Short(mFileName)
End Property

Private Sub Class_Initialize()
  NullChar = Chr(0)
End Sub
```

the properties of the CommonDialog class, to refer to an instance of the common dialog OCX (*not* the VBX). This property procedure then reads the essential properties from the control into its own properties, eliminating the need to code them. Isn't thunking wonderful? The magic of OLE thunking means that the code to implement this feature is very straightforward:

```
Public Property Set CMDLGControl(C As Object)
   Set mCMDLG = C
   FileName = mCMDLG.FileName
   DefaultExt = mCMDLG.DefaultExt
   Filter = mCMDLG.Filter
   FilterIndex = mCMDLG.FilterIndex
   Flags = mCMDLG.Flags
   InitDir = mCMDLG.InitDir
   mhOwner = mCMDLG.Parent.hWnd
End Property
```

Because every object in VB is an OLE object, it's easy to access them externally.

The heart of the class is the Action property, designed to emulate the Action property of the common dialog control. Like most API functions, GetOpenFileName and GetSaveFileName aren't that threatening once you get past the declarations. The top part of the Action property procedure simply assigns values to the OPENFILENAME structure from your properties:

```
OFN.lStructSize = Len(OFN)
   If mhOwner = 0 Then mhOwner = GetActiveWindow()
   OFN.hwndOwner = mhOwner
   OFN.Flags = Flags

   OFN.lpstrDefExt = DefaultExt

   'set the initial directory, otherwise uses current
   Temp = InitDir
   OFN.lpstrInitialDir = Temp
```

Always remember to fill in the size of the structure. If you leave that value blank, the procedure will fail with no discernible error message.

The first interesting part of the code has to do with handling long file names in the 32-bit version of this class. You can pass a short file name to the Dir function (a wrapper around the FindFileFirst and FindFileNext API functions), and it will return the long file name. Therefore, a 16-bit application that has been

VB4

```
VERSION 4.00
Begin VB.Form Form1
  Caption      = "Form1"
  ClientHeight   =  4140
  ClientLeft     =  1545
  ClientTop      =  1830
  ClientWidth  = 6690
  Height         =  4830
  Left        =  1485
  LinkTopic      =  "Form1"
  ScaleHeight  = 4140
  ScaleWidth     =  6690
  Top          = 1200
  Width       =  6810
  Begin VB.TextBox Text1
     Height         =     1815
     Left        =  2040
     MultiLine      =     -1       'True
     ScrollBars     =     3        'Both
     TabIndex    =  0
     Text        =  "TEST.frx":0000
     Top          = 720
     Width       =  2655
  End
  Begin MSComDlg.CommonDialog CD
     Left        =  840
     Top          = 1320
     _version    =  65536
     _extentx    =  847
     _extenty    =  847
     _stockprops  = 0
     defaultext     =     "TXT"
     dialogtitle  = "Test Dialog"
     filename    =  "*.TXT"
     filter      =  "Text Files (*.txt)|*.txt|All _|
        Files (*.*)|*.*"
     filterindex  = 1
     initdir      = "C:\VB4"
  End
  Begin VB.Menu mnuFile
     Caption      = "&File"
     Begin VB.Menu mnuFileOpen
      Caption     = "&Open"
     End
     Begin VB.Menu mnuFileSaveAs
      Caption     = "Save &As..."
     End
  End
End
Attribute VB_Name = "Form1"
Attribute VB_Creatable = False
Attribute VB_Exposed = False

Dim zRealFile As String
Dim zFileName As String

Private Sub Form_Resize()
  Text1.Move 0, 0, ScaleWidth, ScaleHeight
End Sub

Private Sub mnuFileOpen_Click()
  Dim C As Object

  Set C = CreateObject("cmdlgX.CommonDialog")
  Set C.CMDLGControl = CD

  C.Action = 1
  zFileName = C.filename
  If Len(zFileName) Then
     Me.Caption = "Notepad - " & zFileName
     zRealFile = C.ShortFileName
     Open zRealFile For Binary As #1
     Text1.Text = Input(LOF(1), 1)
     Close
  End If
End Sub

Private Sub mnuFileSaveAs_Click()
  Dim C As Object

  Set C = CreateObject("cmdlgX.CommonDialog")
  Set C.CMDLGControl = CD

  C.Action = 2
  zFileName = C.FileTitle
  If Len(zFileName) Then
     Me.Caption = "Notepad - " & zFileName
     zRealFile = C.ShortFileName
     Open zRealFile For Binary As #1
     Put #1, , Text1.Text
     Close
  End If

End Sub
```

LISTING 2 ***Class Access.*** *Use this code to access the CommonDialog class in order to create file-open and file-save dialogs. The code makes the distinction between the FileName property, which it displays, and the ShortFileName property, which it uses to open files.*

working with a short file name such as PROGRA~1.TXT can blindly pass it into the FileName property, and CommonDialog will try to convert it and display it to the user:

```
#If Win32 Then
      If (InStr(Temp, "*") = 0) And InStr(Temp, "?") _
         = 0 Then
         'try to convert it to a long file name
         zTemp = Dir(OFN.lpstrInitialDir & "\" & Temp)
         If Len(zTemp) Then 'we found a match
            Temp = zTemp
         End If
      End If
   #End If
```

This technique is similar to the techniques used by some commercial utilities that allow 16-bit Windows apps to use long file names.

One other thing to notice is that the code uses the String$ function to preallocate string parameters so that the API functions have someplace to return data:

```
OFN.lpstrFile = Temp & String$(255 - Len(Temp), 0)
   OFN.nMaxFile = 255

   OFN.lpstrFileTitle = String$(255, 0)
   OFN.nMaxFileTitle = 255
```

Finally, the Filter property is "filtered" such that the pipes become the null characters expected by the function. This is a classic C language string array. If you retrieve the environment using API calls, it is returned in this null-delimited format as well. A double null denotes the end of the array.

I included the ShortFileName and ShortFileTitle properties as final elements to make this class useful to 16-bit programs. I thought about trying to figure out whether a 16-bit application was calling the code, and considered returning only short file names in that case. But then I figured that even a 16-bit app would want to show the long file name, even if it were using the short one behind the scenes. This would provide the most continuity to the user:

```
Private Function Long2Short(ByVal S As String) As String
   Dim Buff As String
   Dim r As Integer
   #If Win32 Then
      If Dir(S) = "" Then
         Open S For Output As #1
         Close
      End If
      Buff = Space(256)
      r = GetShortPathName(S, Buff, 256)
      Long2Short = Left(Buff, r)
   #Else
      Long2Short = S
   #End If
End Function
```

You implement these properties internally using the GetShortPathName API function, which is pretty straightforward. Passing in the long file name and a buffer will fill the buffer with the shortened version (the one you often see at the DOS prompt).

The beauty of this approach is that you already know how to use this class. You just set properties and use the Action property to bring up the dialog. For example, a 16-bit editor application can display an Explorer-style, file-open dialog and use long file names (see Figure 2). I've included the code to access the CommonDialog class the easy way, using a control for its properties (see Listing 2). Practically no code at all!

Certainly you could do more with this class such as create error handlers, and so on, but it should give you an idea of how to encapsulate system functionality into a compatible class. It also should drive home one of the benefits of OLE Automation: thunking. See you in 60!

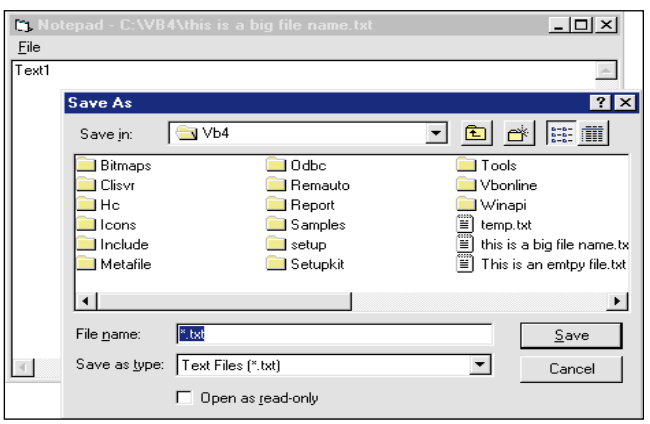

FIGURE 2 ***Yes, Virginia, It Can Really Use Long File Names.*** *Here's an example of a 16-bit Windows application using the Windows 95 Common Dialogs.*