Identify An App's Version

by Chris Barlow

Avoid potential conflicts by creating an app that reads and displays version information for any file.

Click & Retrieve Source CODE!

Once you start writing programs in Visual Basic, you'll also start making changes to these programs. Most developers are like artists working on a painting—it's hard to know when to stop. There always seems to be one more feature you can tweak. As soon as you compile your program for the *second* time, you have a potential problem—multiple versions of your program exist that the user can run. You need to make sure you can always identify which version of your program the user is running.

Visual Basic 4 and 5 make it easy to identify your program by adding a version resource to the EXE file that you can view from Windows Explorer (see Figure 1). Right-click on any application file to display the Properties window and, if the file contains a version resource, you'll see a Version tab that displays information about the file. You can use the Options button on the Make Project window to modify the information about your application. Notice you can set Visual Basic to automatically increment the Revision property each time you compile the application. You can set the Minor and Major properties as you make substantial changes to your application. These values are available to your application as properties of the App object. For example, when your program writes a log file, it can write its version into the log file so you can track problems generated by different versions of your application.

One of the tricks we use at SunOpTech is to add a top-level menu item at the end of the form's menu structure. In the Form Load event, we set its caption to a formatted string of the application's version:

```
Private Sub _
   Form_Load()
mVer.Caption = "v" & _
   FormatVersion()
End Sub

Function _
   FormatVersion() _
   As String
FormatVersion = _
   App.Major & "." & _
   App.Minor & "." & _
   App.Revision
End Function
```

This way, users can easily tell us the application's version simply by looking at the screen. If they print the screen and e-mail or fax it to us, we can see the version that caused the problem.

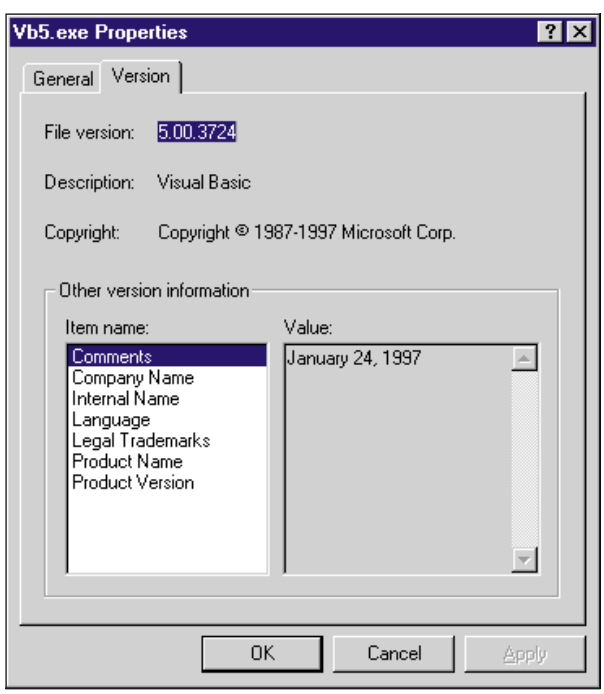
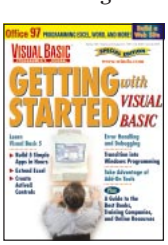


FIGURE 1 ***Windows Explorer File Properties.*** *The Windows Explorer can display the properties of a file, including some of the version resource information. Take advantage of this to ensure you have the correct versions of EXEs and DLLs on a machine.*

Chris Barlow is president and CEO of SunOpTech, a developer of manufacturing decision-support applications, including the ObjectBank, ObjectOrder, and ObjectJob Systems. Chris holds degrees from Harvard Business School and Dartmouth College, where he worked with Drs. Kemeny and Kurtz on the original BASIC language. You can find a number of Chris' articles in VBPJ*'s beginner-oriented sister publication,* Getting Started with Visual Basic*, available on the newsstand. Reach Chris at ChrisB@SunOpTech.com or through SunOpTech's Web server at www.SunOpTech.com.*



READ THE VERSION

Often, however, knowing your own application's version is not enough. You might need to check the version of other application files. With Visual Basic, you easily can write an application that reads and displays the version information from any file. You should encapsulate this functionality into a class in an ActiveX DLL so you can use it for future applications.

A call to the Windows API, GetFileVersionInfo, returns the version resource information from any file. The version resource is relatively complex, but it's flexible enough to contain resource information in multiple foreign languages. To accommodate this variable information, the version resource varies in size. There are several routines in the Windows API to access parts of this information, but we'll use the simpler method of parsing the returned string of resource information.

Begin by launching Visual Basic and creating a new ActiveX DLL project. Add a module to contain these Windows API declarations:

```
Declare Function
GetFileVersionInfoSize Lib _
   "version.dll" Alias _
   "GetFileVersionInfoSizeA" _
   (ByVal lptstrFilename As String, _
   lpdwHandle As Long) As Long
Declare Function GetFileVersionInfo _
   Lib "version.dll" Alias _
   "GetFileVersionInfoA" _
   (ByVal lptstrFilename As String, _
   ByVal dwHandle As Long, ByVal _
   dwLen As Long, lpData As Any) _
   As Long
```

The first function returns the size of the variable-length version resource in the given file. The second function reads the version resource from the file into the lpData buffer. Go to the class module and change the Name property to cFileVer and the Instancing property to Multiuse so other applications can use this class.

The next step in setting up a new class is to define the properties of the class. I usually use the Property Get/Let procedures to validate properties of a class. However, these properties are simply read

VB5

```
Option Explicit
Public sFileName$
Public sFileDateTime$
Public dFileDateTime#
Public lFileLen&
Public bHasVerInfo As Boolean
Public sCompanyName$
Public sFileDescription$
Public sFileVersion$
Public sInternalName$
Public sLegalCopyright$
Public sLegalTrademarks$
Public sOriginalFilename$
Public sProductName$
Public sProductVersion$
Public Major%
Public Minor%
Public Revision%

Public Sub GetFileVersionData()
Dim sVerInfo As String, sTmp As String, lSz&
Dim res As Long, pos As Integer, n As Integer, _
  Hwnd As Long
ClearProps
If Len(sFileName) Then
  lFileLen = FileLen(sFileName)
  dFileDateTime = CDbl(FileDateTime(sFileName))
  sFileDateTime = Format$(dFileDateTime, _
     "dd-mmm-yy hh:nn:ss")
  lSz = GetFileVersionInfoSize(sFileName, Hwnd)
  If lSz Then
     sVerInfo = String$(lSz, 0)
     res = GetFileVersionInfo(ByVal sFileName, _
        0&, ByVal lSz, ByVal sVerInfo)
     If res Then
        bHasVerInfo = True
        sCompanyName = ParseResource(sVerInfo, _
           "CompanyName", 12)
        sFileDescription = ParseResource(sVerInfo, _
           "FileDescription", 16)
        sFileVersion = ParseResource(sVerInfo, _
           "FileVersion", 12)
        sInternalName = ParseResource(sVerInfo, _
           "InternalName", 16)
        sLegalCopyright = ParseResource(sVerInfo, _
           "LegalCopyright", 16)
        sLegalTrademarks = ParseResource(sVerInfo, _
           "LegalTrademarks", 16)
        sOriginalFilename = ParseResource(sVerInfo, _
           "OriginalFilename", 20)
        sProductName = ParseResource(sVerInfo, _
           "ProductName", 12)
        sProductVersion = ParseResource(sVerInfo, _
           "ProductVersion", 16)
        pos = InStr(sFileVersion, ".")
        If pos > 0 Then
           Major = CInt(Left(sFileVersion, pos))
           sTmp = Mid(sFileVersion, pos + 1)
           pos = InStr(sTmp, ".")
           If pos > 0 Then
              Minor = CInt(Left(sTmp, pos))
              Revision = CInt(Mid(sTmp, pos + 1))
           End If
        End If
     End If
  End If
End If
End Sub

Private Function ParseResource(r$, s$, l&) As String
Dim pos&
pos = InStr(r, s)
If pos Then
  ParseResource = TrimNul(Mid$(r, pos + l))
End If
End Function

Private Function TrimNul(s$) As String
Dim pos&
pos = InStr(s, Chr$(0))
If pos Then
  TrimNul = Left(s, pos - 1)
End If
End Function

Private Sub ClearProps()
lFileLen = 0
dFileDateTime = 0
sFileDateTime = ""
bHasVerInfo = False
sCompanyName = ""
sFileDescription = ""
sFileVersion = ""
sInternalName = ""
sLegalCopyright = ""
sOriginalFilename = ""
sProductName = ""
sProductVersion = ""
End Sub
```

LISTING 1 ***FileVer DLL.*** *This ActiveX DLL has a single method to return information from the version resource of a file. Use the ParseResource procedure to search the string buffer for the resource property, then fill the class property.*

from the version resource and the class never uses their values, so you should create them as public variables. The first obvious properties are the fields contained in the version resource:

```
Public sCompanyName$
Public sFileDescription$
Public sFileVersion$
Public sInternalName$
Public sLegalCopyright$
Public sLegalTrademarks$
Public sOriginalFilename$
Public sProductName$
Public sProductVersion$
```

You also might want to add properties to give more complete information to your application. Use the FileName property to pass the fully qualified file. The file's last modified date/time is readily available with Visual Basic's FileDateTime function, but it's convenient to have two properties—one formatted as a string, and one as a double. Similarly, the FileLen function returns the file's length. I also added a Boolean variable to indicate whether the file has a version resource and integer fields for the major, minor, and revision segments of the file's version number:

```
Public sFileName$
Public sFileDateTime$
Public dFileDateTime#
Public lFileLen&
Public bHasVerInfo As Boolean
Public Major%
Public Minor%
Public Revision%
```

This class will have a single method, GetFileVersionData, which will make the calls to the Windows API and fill the class properties. First, clear the properties; then make sure that the sFileName property contains a file. Fill the properties that are valid for any file, even ones that don't have a version resource:

```
Public Sub GetFileVersionData()
Dim sVerInfo As String, sTmp As _
   String, lSz&
Dim res As Long, pos As Integer, n As _
   Integer, Hwnd As Long
ClearProps
If Len(sFileName) Then
   lFileLen = FileLen(sFileName)
   dFileDateTime = CDbl_
      (FileDateTime(sFileName))
   sFileDateTime = Format$_
      (dFileDateTime, _
      "dd-mmm-yy hh:nn:ss")
```

Next, find out the size of the version resource in the file. If this call returns zero, the file doesn't have a valid version resource. Otherwise, you can create a string variable the size of the version resource and pass it as an argument in the next API call:

```
   lSz = _
      GetFileVersionInfoSize_
      (sFileName, Hwnd)
   If lSz Then
      sVerInfo = String$(lSz, 0)
      res = GetFileVersionInfo(ByVal _
         sFileName, 0&, ByVal lSz, _
         ByVal sVerInfo)
```

If this call returns a nonzero value, the version resource has been loaded into the string buffer. The data is a typical C structure format with null characters terminating the strings. Use the ParseResource procedure to search the string buffer for the resource property, then fill the class property (see Listing 1):

```
If res Then
   bHasVerInfo = True
   sCompanyName = ParseResource_
      (sVerInfo, "CompanyName", 12)
   sFileDescription = ParseResource_
      (sVerInfo, "FileDescription", 16)
   sFileVersion = ParseResource_
      (sVerInfo, "FileVersion", 12)
   sInternalName = ParseResource_
      (sVerInfo, "InternalName", 16)
   LegalCopyright = ParseResource_
      (sVerInfo, "LegalCopyright", 16)
   sLegalTrademarks = ParseResource_
      (sVerInfo, "LegalTrademarks", 16)
   sOriginalFilename = ParseResource_
      (sVerInfo, "OriginalFilename", _
      20)
   sProductName = ParseResource_
      (sVerInfo, "ProductName", 12)
   sProductVersion = ParseResource_
      (sVerInfo, "ProductVersion", 16)
```

Finally, parse the sFileVersion property to fill the integer version properties:

```
         pos = InStr(sFileVersion, _
            ".")
         If pos > 0 Then
            Major = CInt(Left_
               (sFileVersion, pos))
            sTmp = Mid(sFileVersion, _
               pos + 1)
            pos = InStr(sTmp, ".")
            If pos > 0 Then
               Minor = CInt(Left_
                  (sTmp, pos))
               Revision = CInt(Mid_
                  (sTmp, pos + 1))
            End If
         End If
      End If
   End If
End If
End Sub
```

TEST HARNESS

Now that your class is complete, you need a small test application to call the method and display the results (download Listing 2 from The Development Exchange; see the Code Online box for details). Add a standard EXE project to your project group, and place a list-box control and a common dialog control on the form. Add a File menu with Open and Exit menu items. In the Open event, add this code to show the common dialog and call a procedure to show the version resource information:

```
Private Sub mOpen_Click()
CommonDialog1.ShowOpen
ShowFileVersionData _
   CommonDialog1.filename
End Sub
```

In the ShowFileVersionData procedure, dimension an instance of your cFileVer class and set the sFileName property. Then call the GetFileVersionData method and fill the list-box control with the properties of the class.

Take a look at different files with your test harness. You can download the files for both the class DLL and the test harness from the Premier Level of The Development Exchange (see the Code Online box at the end of the column for details). Now you can use your new class to get the version information from any file and make sure your application works with the proper versions. Next month, learn how to use this class to create a program that always launches the latest version of your applications. ☒

Code Online

You can find all the code published in this issue of VBPJ *on The Development Exchange (DevX) at http://www.windx.com. All the listings and associated files essential to the articles are available for free to Registered members of DevX, in one ZIP file. This ZIP file is also posted in the Magazine Library of the* VBPJ *Forum on CompuServe. DevX Premier Club members ($20 for six months) can get each article's listings in a separate file, as well as additional code and utilities for selected articles, plus archives of all code ever published in* VBPJ *and* Microsoft Interactive Developer *magazines.*

Identify An App's Version Locator+ Codes

Listings ZIP file, including Listing 2, which was excluded for space reasons (free Registered Level): VBPJ0897

✪ *Listings for this article plus the source code for the ActiveX DLL and the test harness application (subscriber Premier Level): GS0897P*