November, 2000

ADVISOR Answers

Application Version Information

VFP 6.0/5.0

Q: How do you find out the version number of a compiled VFP .EXE file at runtime? I'm referring to the version number from the Build/Version screen of the project manager, not simply the version number of VFP. I've used the following, which returns the version number in the fourth array element but isn't there a better way?

AGetFileVersion(aFileInfo, PROGRAM() + '.EXE')

–Steve Koch, Honolulu, HI (via Advisor.com)

A: In fact, the AGetFileVersion() function was added in VFP 6 specifically for this purpose. It provides version information about any .EXE or .DLL that has the information stored in it. It's the same information you can find by right clicking on the file in Windows Explorer and choosing Properties, then going to the Version page of the Properties dialog.

AGetFileVersion() actually returns 15 different pieces of information about the file. You pass an array and the file name (including the path, if necessary). If the file exists and has version information, the array is created, if it doesn't exist, or resized if it does, and filled with the version information.

The function returns 15 when it's successful and 0 when it's unsuccessful. In code, however, it's best to test for 0 or a non-zero value, since it's possible that the number of elements returned and thus, the value returned may change in future versions of VFP.

What are all those array elements and how do you specify them? Table 1 shows the contents of the array created by AGetFileVersion(). Some of the elements there may seem to be redundant. What's the difference between "internal filename," "original filename," and "product name?" What's the difference between "file version" and "product version?" What's a "private build" or a "special build?" That depends what company you ask.

Table 1. File Version Information – The AGetFileVersion() function provides the information that's in the Properties dialog for .EXE and .DLL files and a whole lot more.

Element	Contents
1	File comments
2	Company Name
3	File Description
4	File Version
5	Internal Filename
6	Copyright
7	Trademarks
8	Original Filename
9	Private Build
10	Product Name
11	Product Version
12	Special Build
13	Does this file register itself for OLE? Contains "OLESelfRegister" if so; empty otherwise.
14	Language
15	Translation code

Microsoft defined this format for file information and has its own definitions for these terms, but not every other company applies them the same way. It's a pretty safe bet that the fourth item in the array will give you the version number of a file and that the tenth item will give the public name, the one that appears in the splash screen. In Microsoft's view, the fifth element, internal filename, doesn't include the file's extension, while the eighth element, does. However, when I

tested applications from other vendors, sometimes the fifth element contained the file extension and sometimes it didn't.

Several of the items simply contain whatever text has been stored in them. They include the first element (file comments), the second element (company name), the third element (file description), the sixth element (copyright) and the seventh element (trademarks).

By now, you're probably wondering how you get all this information into your executable. There are two ways to do in VFP 6. One is interactive, while the other is programmatic.

Interactively, the Project Manager's Build dialog includes a Version button. When you click it, the dialog in Figure 1 appears. You can specify the version number, as well as the various textual items for the file.

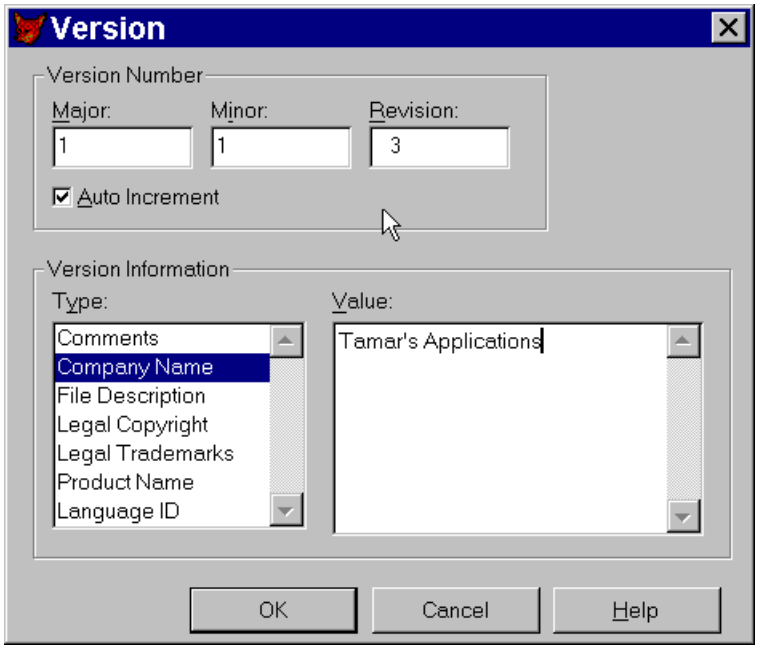

Figure 1. Adding version information–This dialog, accessible through the Project Manager's Build dialog, lets you specify the version number and other information about your project that can then be retrieved using AGetFileVersion().

If you check the Auto-Increment checkbox, the last part of the Version number, labeled Revision in Figure 1, will go up by 1 each time you build the project, so you don't have to remember to do it yourself.

It's also possible to specify the project's version information programmatically by using the Project object. Whenever you open a project in VFP 6 and later, a Project object is created. You can access

it through the _VFP system variable. The Project object has properties with names like VersionComments, VersionCopyright and VersionTrademarks. You can set these properties. Then, when you build the project, those values will be built into the executable just as it you'd entered those values in the Version dialog. For example, to set the properties of a project called FoxIsGreat, you could do the following:

```
MODIFY PROJECT FoxIsGreat NOSHOW
oProject = _VFP.ActiveProject
WITH oProject
   .VersionComments = "This is the latest and " + ;
      "greatest version of the FoxIsGreat product!"
   .VersionCompany = "Tamar's Applications"
   .VersionDescription = "FoxIsGreat 3.0"
   .VersionNumber = "3.0.0203"
   .AutoIncrement = .T.
   .VersionCopyright = "Copyright 2000, Tamar's Applications"
   .VersionProduct = "FoxIsGreat"
ENDWITH
```

It's worth noting that you specify the language as a numeric value (that is VersionLanguage is numeric), but the 14th element of the array returned by AGetFileVersion() indicates the language as a character string. In the Version dialog, you can specify the language using either its name or its numeric id.

Versions before VFP 6 had neither the AGetFileVersion() function nor the Project object. In VFP 5, you could still specify version information interactively using the Version dialog shown in Figure 1. Beginning in VFP 5, the FoxTools library includes a function called GetFileVersion() that lets you extract file version information from .EXE and .DLL files. It accepts parameters in the opposite order of AGetFileVersion() – that is, it takes the file name first, then the array. GetFileVersion() returns only 12 items (the last 3 in Table 1 are omitted). You must create the array with 12 elements and pass it by reference to the function. GetFileVersion() returns 0 if it's successful and –1, if it fails.

In VFP 3, there's no easy way to specify version information and it's necessary to use API calls to retrieve it.

Now back to your original question–getting the version information from the application that you're running. AGetFileVersion() is the way to go. However, the example you offer won't work because PROGRAM() returns the program you're executing at the moment, which may or may not be the one at the top of the program chain. It

also isn't the .EXE, but the program, form, method or menu that's running.

You need to make sure you pass the name of the .EXE file. To do that, use SYS(16, 1), which returns the name of the executable (or application). However, depending on what you choose for your main program, even SYS(16,1) may return something other than the .EXE or .APP file. So, your best bet is to grab that information immediately, in the main routine itself. For an application that uses a form as the main program, call SYS(16,1) in the Load method and save the information in an application property. For a menu-based main program, do it in the Setup code. In any application, you can save the file name of the .EXE or .APP with something like this, where goApp is the variable that references the application object:

```
goApp.EXEFile = SYS(16,1)
```

Once you've stored the file name, to determine file version information from anywhere in your application, use code like this:

```
IF AGetFileVersion(aVersionInfo, goApp.EXEFile) > 0
   * Proceed
ELSE
   * This EXE doesn't have version info
ENDIF
```

–Tamar