December, 1996

Advisor Answers

FoxPro 2.x and Visual FoxPro

Q: Are you aware of any method for detecting whether a FoxPro FPW2.6 application is running in Runtime mode or Development mode?

There are certain features (like error handling, security rights, etc.) that slow down my development process but must be in place when I distribute my application to my end users.

I've looked high and low and haven't been able to find anything, although I do seem to recall reading something somewhere about detecting the FPW running state (Development /Runtime).

–Chris D'Arrigo (via CompuServe)

A: In FoxPro 2.x, determining runtime vs. development is slightly roundabout. In Visual FoxPro, it's quite simple. In both cases, the VERSION() function is the key.

As its name suggests VERSION() tells you about the version of FoxPro in use. Called with no parameters, it returns the name of the version you're running. For example, in FoxPro 2.6a for Windows, VERS() returns the string "FoxPro 2.6a for Windows". The extended version of FoxPro 2.6a for DOS returns simply "FoxPro 2.6a (X)". In Visual FoxPro, the return value includes the "build" number. For example, Visual FoxPro 3.0b returns "Visual FoxPro 03.00.00.0711 for Windows" when you call VERS(). The 711 at the end of the version number indicates the internal "build" of the version. (The original version of Visual FoxPro 3.0 was build 596.)

When you pass the number 1 to VERSION(), it gets more informative and returns the date and time of the build, as well as your product id.

When you use a runtime version, the return value of VERSION() changes. An executable (compact or standalone) running with the support libraries in FoxPro for DOS 2.6a returns "FoxPro 2.6a EXE Support Library" for VERS() and the same string plus date and time and the word "Product" for VERS(1). Similarly, in FoxPro 2.6a for Windows, VERS() returns "FoxPro 2.6a EXE Support Library for Windows".

The difference in the return values means you can test VERS() for the string "EXE" to determine if you're in a development or a runtime version of FoxPro. A line like this early in your application will do the trick:

```
lDevelopment = NOT ("EXE"$VERSION())
```

Then you can check lDevelopment throughout your application to determine whether you're in development mode or not.

Visual FoxPro distinguishes the development and runtime versions in the same way (with a different return value for VERSION()); both .EXEs and .DLLs built with Visual

FoxPro return the runtime version of the string. However, there's a more direct approach. In Visual FoxPro, the VERSION() function accepts some alternate parameters. Passing 2 lets you determine not just whether you're in development or runtime, but for a development version whether you're using the Standard or the Professional edition. So, in a VFP application, you'd use something like:

```
lDevelopment = VERS(2)<>0
```

Note that the return value of VERSION(2) is numeric while all other return values for VERSION() are character. Again, both .EXEs and .DLLs return 0 for VERSION(2).

Passing 3 to VERSION() lets you determine which localized version of VFP you're using - "00" indicates English while other values identify other languages - see the online help for a complete list of values.

VFP 5 also introduces a Version property of the application object that might be useful when using VFP as an OLE Server. However, it returns only the version number (5.0 in the beta version I tested) and none of the additional information. However, the new StartMode property of Application provides information about how the VFP session was started. It distinguishes between a normal VFP session and several types of OLE sessions.

I've also occasionally wanted to know, in the development version, whether I was running an .APP or an .EXE. One way to figure that out is to test SYS(16) (which returns the full name of the currently executing program) and parse out the file extension. Use SYS(16,0) to ensure that you're looking at the main program of your application if you don't check it right away. Some functions built into the FoxTools (in the Windows versions) and FPath (in the DOS version) libraries make parsing file names easy. In this case the JustExt function picks out the extension from a full file name, so code like this does the trick:

```
SET LIBRARY TO FoxTools
* or, in FP/D:
* SET LIBRARY TO FPath
IF JustExt(SYS(16))="APP"
   lIsItApp = .T.
ELSE
   lIsItApp = .F.
ENDIF
```

Finally, VFP 5 introduces the ability to brand your applications with its own version number divided into three parts, as well as to specify copyright and other information for the .EXE. The new GetFileVersion() function in FoxTools lets you retrieve the information you've stored. To use GetFileVersion(), you need to declare an array with 12 items. Pass the .EXE file name as the first parameter and the array by reference as the second, like this:

```
DIMENSION aVersionInfo(12)
lnRetVal = GetFileVersion("MyCool.EXE",@aVersionInfo)
```

and the array is filled with the information you specified. The function returns 0 when it's successful and -1 when it fails (for example, if the file you pass is not an .EXE or .DLL). GetFileVersion() works on a lot of .EXE and .DLL files, but not all of them, not even

every .EXE from Microsoft. (It failed on EXCEL.EXE, the Excel 7 executable, but happily identified the Office 95 versions of MSACCESS.EXE and WINWORD.EXE.)

–Tamar