January, 2004

Advisor Answers

Getting a directory listing

VFP 8/7/6

Q: I'm getting different results running the DIR command from within VFP and from a command window. When I use VFP's DIR command, I get a list of filenames in two or more columns. When I issue:

```
! DIR *.zip > zipfiles.txt
```

the format for the files in the resulting listing looks like this:

```
RF001    ZIP  14056982 16/09/03   22:19
```

When I open a command prompt from Windows and issue the same command:

```
DIR *.zip > zipfiles.txt
```

the listing for a file looks like this:

```
09/16/2003  10:19 PM        14,056,982 RF001.zip
```

I need to get an accurate list of filenames that I can put into a cursor for processing. Some of the directories involved contain more than 100,000 files, so I can't use ADIR().

What's causing the difference in the results from the DIR command? Is there a better way to gather information about files in a directory that contains many files?

–Erez Miller (via Advisor.COM)

A: Let me start with your first question. Why are you getting different results from the three different ways of listing the directory contents? The VFP DIR command (actually short for DIRECTORY) is a synonym for the LIST FILES command. Both are built into the FoxPro engine and produce output in a format similar to the "wide" format you get when passing the /W switch to the DOS DIR command. There's no way to control the output format.

You would expect RUN DIR (or the identical ! DIR) to produce the same results as issuing DIR from a command prompt. A little exploration turns up the /N switch for the DIR command, which is described as using the "New long list format where filenames are on the right." That's the format you're seeing at the command prompt, while RUN DIR is using the older traditional format. You can see the older format at the command prompt by issuing the command like this:

```
DIR /-N *.zip > zipfiles.txt
```

In addition to the difference in the order of the columns, the older format doesn't support long file names. Files are shown using their short names.

However, I suspect you actually prefer the long format. Unfortunately, I can't get the /N switch to work via the RUN command, like this:

```
! DIR /N *.zip > zipfiles.txt
```

The problem is that there are actually two different versions of the command processor, COMMAND.COM and CMD.EXE. COMMAND.COM, which is what VFP normally uses when you issue the RUN command, doesn't offer the /N switch. You have a couple of options if you want to use the DOS DIR command.

The first is to change FOXRUN.PIF, the file that controls the FoxPro RUN command. Find the file in the VFP home directory, right-click and choose Properties. On the Program page, change the Cmd line entry to CMD.EXE.

The second alternative is to call on CMD.EXE directly with code like:

```
! cmd.exe /C DIR /n *.zip >zipfiles.txt
```

However, neither of these solutions work in Windows 9x. In addition, both solutions suffer from the problem using RUN generally has; there's an ugly DOS window displayed along the way. Fortunately, there are a number of other solutions to the problem.

As you note, ADIR() isn't a choice for you because of the number of files you're dealing with. Arrays are limited to 65,000 total elements; with 5 columns per item, that means it works only for directories with 13,000 files or fewer. (This problem will go away in VFP 9, where the limit on arrays will be eliminated.)

One option is to use a rather obscure VFP command. (In fact, so obscure that I'd forgotten about it until Christof reminded me.)

SYS(2000) lets you go through the files in a directory one by one. The syntax is:

```
cFileName = SYS(2000, cFileSpec [, 1])
```

The first time you call it, omit the third parameter and it returns the first matching file. On subsequent calls, pass the third parameter and it returns the next matching file. Here's a function (FindFiles2000.PRG on this month's Professional Resource CD) that fills a cursor with the specified files. You pass the directory and file spec, and the function returns the number of files found.

```
* Find all files in a specified directory
* that match a filespec, using SYS(2000).
LPARAMETERS cFolder, cFileSpec

LOCAL cSearchName, cFile

CREATE CURSOR FileList (mFileName M)

cSearchName = FORCEPATH(cFileSpec, cFolder)

cFile = SYS(2000,cSearchName)

DO WHILE NOT EMPTY(cFile)
   * Extract file name
   INSERT INTO FileList VALUES (m.cFile)

   cFile = SYS(2000,cSearchName,1)
ENDDO
nReturn = RECCOUNT("FileList")

RETURN nReturn
```

In addition, there are two possible approaches using tools provided by Windows. The first is the Windows Scripting Host, and the second is to go right to the Windows API. I'll show you both solutions and then explain why I recommend the API version, if you're going outside the native language.

The FileSystemObject of the Windows Scripting Host (WSH) gives you access to the drives, folders and files on a computer. While it doesn't offer a way to search for files matching a particular pattern, you can get a list of all the files in a folder and then use VFP's LIKE() function to check whether they match the desired filespec. Here's my code to populate a cursor with the names of all the files in a folder that match a specified pattern. It returns the number of files found; if the specified directory doesn't exist (or something else goes wrong), it returns –1. (It uses VFP 8's TRY-CATCH construct to handle any problems. To use it in VFP 7 or earlier, you'll need to add some error handling.) You'll find it on this month's Professional Resource CD as FindFilesWSH.PRG:

```
* Find all files in a specified directory
* that match a filespec, using Scripting.
LPARAMETERS cFolder, cFileSpec

LOCAL oFSO as Scripting.FileSystemObject
LOCAL oFolder AS Scripting.Folder
LOCAL oFiles as Scripting.Files
LOCAL oFile as Scripting.File
LOCAL nReturn

CREATE CURSOR FileList (mFileName M)
TRY
   oFSO = CREATEOBJECT("Scripting.FileSystemObject")
   oFolder = oFSO.GetFolder(cFolder)
   oFiles = oFolder.Files
   FOR EACH oFile IN oFiles
      IF LIKE(cFileSpec, oFile.Name)
         * Found one. Add it.
         * If full path is needed, use oFile.Path instead
         INSERT INTO FileList VALUES (oFile.Name)
      ENDIF
   ENDFOR
   nReturn = RECCOUNT("FileList")
CATCH
   nReturn = -1
ENDTRY

RETURN nReturn
```

The Windows API approach to looking inside a directory is like the one used by SYS(2000). You ask for the first file matching a pattern and then loop through the rest of the matching files. Two API functions are needed, FindFirstFile and FindNextFile. Both functions fill a string parameter passed by reference with a great deal of information about the file. The code here simply parses out the filename. Here's the API version of the function (on this month's PRD as FindFilesAPI.PRG):

```
* Find all files in a specified directory
* that match a filespec, using the Win API.
LPARAMETERS cFolder, cFileSpec

#DEFINE FIND_DATA_SIZE 318
#DEFINE START_NAME 45
#DEFINE MAX_PATH_LEN 260

DECLARE INTEGER FindFirstFile IN kernel32;
    STRING   lpFileName,;
    STRING @ lpFindFileData
```

```
DECLARE INTEGER FindNextFile IN kernel32;
    INTEGER   hFindFile,;
    STRING  @ lpFindFileData

LOCAL cFileData, hFile, lReturn, cSearchName
LOCAL cFile

CREATE CURSOR FileList (mFileName M)

cSearchName = FORCEPATH(cFileSpec, cFolder)
cFileData = SPACE(FIND_DATA_SIZE)

hFile = FindFirstFile(cSearchName, @cFileData)
hFirst = hFile
DO WHILE hFile > 0
   * Extract file name
   cFile = SUBSTR(cFileData, START_NAME, ;
                  MAX_PATH_LEN) + CHR(0)
   cFile = LEFT(cFile, AT(CHR(0), cFile)-1)
   INSERT INTO FileList VALUES (m.cFile)

   hFile = FindNextFile( hFirst, @cFileData)
ENDDO
nReturn = RECCOUNT("FileList")

RETURN nReturn
```

There are a couple of differences between the two versions. First, the API version returns 0 if the directory doesn't exist rather than –1 (though you could change that by testing for hFile=-1 before the DO WHILE). More importantly, the API version sees hidden files, which the WSH and SYS(2000) versions do not.

Those differences aside, there are two important reasons why I recommend the API version. The first is that the WSH may not be available on every computer that runs your application. Because malicious code can use it, some system administrators don't allow the WSH on their networks.

The second reason is even more compelling. The API version is much, much faster. I tested the speed of both versions by looking for files in my Windows SYSTEM32 directory (where there are about 2300 files). The API version ran 20 to 30 times faster. That said, the SYS(2000) version was slightly faster than the API version, so is generally your best choice, unless you need to include hidden files, or have to preserve the case of file names.

–Tamar