October, 1999

Advisor Answers

Finding the Most Recent Files

Visual FoxPro 6.0, 5.0 and 3.0 and FoxPro 2.x

Q: I work primarily in FoxPro 2.6. There are times when I would like to compare the names and date stamps of the files in one subdirectory with those in another subdirectory. We bring back files from client sites and take files to client sites and we make backups. It would be nice to be able to tell which files are the most recent, without having to call up each list in date order or copying files with Windows Explorer and then answering "No" a bunch of times. I know there are programs that use this feature, but I would like to know how to write one or obtain an add-on to use to do this.

—Alvin Fisher (via Advisor.COM)

A: This being FoxPro, there are several ways to do what you want. In fact, I came up with four approaches, two of which work in FoxPro 2.x as well as VFP. Let's start with those two.

First, let's lay out the problem. The function receives three parameters: the two paths, plus a file specification in the usual format (like "*.PRG"). Each version of the function produces as output a cursor with one row for each unique filename found in either directory. The cursor has three columns – the filename, the path where the most recent or only version of that file was found, and a logical value indicating whether the file exists in only one of the specified directories or both. The function returns .F. if there's a problem with the parameters and .T. otherwise.

Both of the FoxPro 2.x-compatible versions depend on the ADIR() function. ADIR() accepts a file specification and fills an array with all matching files. For example:

```
nFiles = ADIR(aFileList, "*.prg")
```

populates aFileList with the list of .PRG files in the current directory, creating or resizing the array as needed.

The resulting array has five columns. The first contains the name of the file, while the third and fourth contain the file date and time, respectively. So, one call to ADIR() for each directory involved provides all the information we need.

Once we have an array of files from each directory, we need to compare them. I tried two approaches for performing the comparison and found that one way was somewhat faster than the other (and also faster than the two techniques that don't use ADIR()). The slower technique loops through the first array, and, for each item, searches the second for a match. If no match is found, the item from the first array is moved to the results. When there is a matching item in the second list, the dates and times are compared and the newer item is put into the cursor. After the loop through the first list is finished, all unmatched items from the second list are copied to the result.

The second method sorts the lists, then walks through them in parallel, each time picking up the item that comes first alphabetically and putting it into the result set. When the items match, the dates and times are compared and the newer one goes into the result. After one list is exhausted, the rest of the other list is copied to the result. Here's the FoxPro 2.x-compatible version of the code for it. The VFP-only version isn't very different, but takes advantage of a few of the things that have changed in the interim.

```
* Compare files in two directories using arrays.
* This version walks through the two arrays in parallel.
* Produces a cursor containing a list of files
* showing the most recent version of matching
* files and indicating files that exist in only
* one directory.
* FoxPro 2.x-compatible version

PARAMETERS cFirstPath, cSecondPath, cFileSpec

IF Type("cFirstPath") <> "C" OR ;
   Type("cSecondPath") <> "C"

   WAIT WINDOW "Must pass two paths"
   RETURN .F.
ENDIF

IF Type("cFileSpec") <> "C"
   WAIT WINDOW "Must pass filespec"
   RETURN .F.
ENDIF

* Create a cursor to hold the results
* Use a memo for filename to support long filenames
* (Long file names are an issue only in VFP)
CREATE CURSOR CurrFls (mFileName M, mPath M, lOnlyOne L)

* Get the lists
* Need to load FoxTools or FPath
lLoadedLib = .F.
cOldLibr = SET("LIBRARY")

IF _DOS
   IF NOT ("FPATH"$cOldLibr)
      SET LIBRARY TO (HOME() + "FPath") ADDITIVE
      lLoadedLib = .T.
   ENDIF
ELSE
   IF NOT ("FOXTOOLS"$cOldLibr)
      SET LIBRARY TO (HOME() + "FoxTools") ADDITIVE
      lLoadedLib = .T.
   ENDIF
ENDIF

nFirstCount = ADIR(aFirstList, ADDBS(cFirstPath) + cFileSpec)
nSecondCount = ADIR(aSecondList, ;
                    ADDBS(cSecondPath) + cFileSpec)

* If either directory is empty, put everything
* in the other into the results
DO CASE
```

```
CASE nFirstCount = 0 AND nSecondCount = 0
   WAIT WINDOW "No files in either directory"

CASE nFirstCount = 0
   FOR nItem = 1 TO nSecondCount
      INSERT INTO CurrFls VALUES ;
         (aSecondList[ nItem, 1], ;
          cSecondPath, .T.)
   ENDFOR

CASE nSecondCount = 0
   FOR nItem = 1 TO nFirstCount
      INSERT INTO CurrFls VALUES ;
         (aFirstList[ nItem, 1], ;
          cFirstPath, .T.)
   ENDFOR

OTHERWISE
   * There are files in both directories.
   * Go through the lists, matching up files.
   * Put the later file in the result.
   * When a file is unmatched, put it into the result.

   * First, sort both arrays on column 1.
   =ASORT(aFirstList)
   =ASORT(aSecondList)

   lFinished = .F.
   nFirstRow = 1
   nSecondRow = 1

   * Need to set EXACT ON in case one file name
   * is nested in another
   cOldExact = SET("EXACT")
   SET EXACT ON

   DO WHILE NOT lFinished

      * Compare names
      DO CASE
      CASE aFirstList[ nFirstRow, 1] < ;
           aSecondList[ nSecondRow, 1]
         lOnlyOne = .T.
         lUseFirst = .T.

      CASE aFirstList[ nFirstRow, 1] > ;
           aSecondList[ nSecondRow, 1]
         lOnlyOne = .T.
         lUseFirst = .F.

      OTHERWISE
         * There's a match, so compare date and time
         lOnlyOne = .F.
         DO CASE
         CASE aFirstList[ nFirstRow, 3] > ;
              aSecondList[ nSecondRow, 3]
            lUseFirst = .T.
         CASE aSecondList[ nSecondRow, 3] > ;
              aFirstList[ nFirstRow, 3]
            lUseFirst = .F.
         OTHERWISE
            * Same date. Compare times

            IF aFirstList[ nFirstRow, 4] > ;
               aSecondList[ nSecondRow, 4]
               lUseFirst = .T.
            ELSE
               lUseFirst = .F.
            ENDIF
         ENDCASE

      ENDCASE

      IF lUseFirst
         INSERT INTO CurrFls VALUES ;
            (aFirstList[ nFirstRow, 1], ;
             cFirstPath, m.lOnlyOne)

      ELSE
         INSERT INTO CurrFls VALUES ;
            (aSecondList[ nSecondRow, 1], ;
             cSecondPath, m.lOnlyOne)

      ENDIF

      * Move the array pointer(s).
      DO CASE
      CASE NOT lOnlyOne
         nFirstRow = nFirstRow + 1
         nSecondRow = nSecondRow + 1
      CASE lUseFirst
         nFirstRow = nFirstRow + 1
      CASE NOT lUseFirst
         nSecondRow = nSecondRow + 1
      ENDCASE

      * Check whether we're finished either array
      IF nFirstRow > nFirstCount
         lFinished = .T.
      ENDIF
      IF nSecondRow > nSecondCount
         lFinished = .T.
      ENDIF

   ENDDO

   * Now pick up extras from unfinished list
   IF nFirstRow <= nFirstCount
      FOR nItem = nFirstRow  TO nFirstCount
         INSERT INTO CurrFls VALUES ;
            (aFirstList[ nItem, 1], ;
             cFirstPath, .T.)
      ENDFOR
   ENDIF

   IF nSecondRow <= nSecondCount
      FOR nItem = nSecondRow TO nSecondCount
         INSERT INTO CurrFls VALUES ;
            (aSecondList[ nItem, 1], ;
             cSecondPath, .T.)
      ENDFOR
   ENDIF
```

```
   * Restore EXACT setting
   SET EXACT &cOldExact

ENDCASE

RETURN .T.
```

The third attempt at the problem also starts with ADIR(). This version then dumps the array results into a pair of cursors and uses SQL SELECT to pull out the newest versions. Here's the key piece of code from that approach:

```
SELECT FirstList.cFileName, cFirstPath, ;
       IIF(ISNULL(SecondList.cFileName), .T., .F.) ;
   FROM FirstList ;
      LEFT JOIN SecondList ;
        ON FirstList.cFileName = SecondList.cFileName ;
   WHERE FirstList.dDate > SecondList.dDate ;
      OR (FirstList.dDate = SecondList.dDate ;
          AND FirstList.cTime >= SecondList.cTime) ;
UNION ;
SELECT SecondList.cFileName, cSecondPath, ;
       IIF(ISNULL(FirstList.cFileName), .T., .F.) ;
   FROM SecondList ;
      LEFT JOIN FirstList ;
        ON SecondList.cFileName = FirstList.cFileName ;
   WHERE SecondList.dDate > FirstList.dDate ;
      OR (SecondList.dDate = FirstList.dDate ;
          AND SecondList.cTime > FirstList.cTime) ;
   INTO CURSOR CurrentFiles
```

The first query gathers all files that either appear only in the first directory or are newer there, while the second does the same for the second directory. Because of the outer join, this version doesn't work in FoxPro 2.x, though it's likely you can come up with a version that does by using additional UNIONs. I didn't pursue it since the array-based approaches were faster.

The last technique I tried uses the file system object described in Gene Sally's article in the May '99 FoxPro Advisor. Rather than creating either arrays or cursor, it uses the Files collections accessible from that object. The main processing loop is quite similar to the first array technique – it goes through one Files collection, checking each file to see whether it exists in the other collection and figuring out what to do. Unfortunately, this approach tested out about an order of magnitude slower than any of the native versions.

You'll find all six functions comparison functions (the four approaches in VFP code, plus the two 2.x versions) on this month's Professional Resource CD.

—Tamar