Issue Date: FoxTalk September 1996

3.0 or Not 3.0? That's the Question!

Art Bergquist
abergquist@visionpace.com

How do you tell if a table is a Visual FoxPro table or not? That's easy! You display the value of SYS(2029) to return either a 48 (for a "Visual FoxPro table with or without a memo field" message) or some other number (see VFP's online help for other possible values).

What if you want to determine if a table is a Visual FoxPro table but the table isn't open? SYS(2029) returns a 0 ("No table open") in that case, so that isn't very helpful. You could open the table first, but what if you need to know *without* first opening the table.

A technical way of doing this is to go to MS-DOS and use DEBUG to inspect the table's contents (at the DEBUG dash (-) prompt type 'd100' and then type 'q' to Quit). A Visual FoxPro table has a character zero (0) in the first byte (byte 0). Non-VFP tables (and other DOS files in general, for that matter) have a non-zero value in their first byte.

The easier way is to use the following function (I've aptly named it VFPTABLE), which simply reads the first byte (byte 0) and returns .T. if it's a character zero (0) and .F. if isn't. The function thoroughly validates the input parameters by ensuring that a non-empty character string is passed, that this string represents a DOS file that actually exists, and, finally, that the file can be opened (in testing VFPTABLE from the Visual FoxPro Command window, for example, C:\VFP\VFP.EXE couldn't be opened by FOPEN):

```
* Program Name: VFPTABLE.PRG
*       Author: Art Bergquist
*         Date: May 11, 1996
* Purpose: Determine if the specified file is a Visual
*          FoxPro table or not
*   Syntax: m.lIsVFP = vfptable("mytable.dbf")
*   Input: tcFileName = Name of the DOS file to check
*  Output: .T. = the DOS file is a Visual FoxPro table
*          .F. = the DOS file is NOT a Visual FoxPro table

PARAMETERS tcFileName

#INCLUDE foxpro.h

PRIVATE ALL LIKE l_*
* name of the currently executing program
l_Program = PROGRAM()

DO CASE
  CASE PARAMETERS() = 0
    DO ErrorMsg WITH l_Program, ;
       'You must supply a filename.'
    RETURN .F.

   CASE TYPE('tcFileName') # 'C'
    DO ErrorMsg WITH l_Program, ;
       'You must supply a *character* filename.'
    RETURN .F.

  CASE EMPTY(tcFileName)
    DO ErrorMsg WITH l_Program, ;
       'You must supply a non-empty filename.'
    RETURN .F.

  CASE NOT FILE(tcFileName)
    DO ErrorMsg WITH l_Program, ;
       UPPER(tcFileName) + ' does not exist!'
    RETURN .F.

  OTHERWISE
    l_FileName = UPPER(tcFileName)
ENDCASE

* open the file (in read/only mode)
l_FileHandle = FOPEN(l_FileName, 0)
IF l_FileHandle = -1
```

```
  DO ErrorMsg WITH l_Program, ;
     l_FileName + ' could not be opened (by FOPEN).'
  RETURN .F.
ENDIF
* Read the first byte
l_FirstByte  = FREAD(l_FileHandle, 1)
* close the first file
= FCLOSE(l_FileHandle)

* First Byte = '0' if it's a Visual FoxPro table
* non-0 if it's not a VFP table.
l_Return = (l_FirstByte = '0')

RETURN l_Return

*-------------------------------------------------
* ErrorMsg: Display an error message
*-------------------------------------------------
PROCEDURE ErrorMsg
PARAMETERS tcProgram, tcErrorMsg
?? CHR(7)
=MessageBox(tcErrorMsg, MB_OK + MB_ICONSTOP, ;
  'Error in ' + tcProgram + ' function')
RETURN
```

In what situation would you need to use such a function? I'm glad you asked. The following program can be used to import/add FoxPro tables (.DBFs) to a Visual FoxPro database (.DBC). If the tables you are adding are Visual FoxPro free tables, then no problem. If any table is a FoxPro 2.x-style table, however, a dialog box with the following message will appear:

```
File '<dbf_name>.DBF' will be updated to a new file format,
making it unreadable to previous versions of FoxPro.

This can be reversed with the 'COPY TO' command, and the
old table will be saved as a .BAK file.  Continue?

            << Yes >>        < No >
```

If you don't mind running this program interactively (in other words, answering Yes anytime the program is about to add a FoxPro 2.x-style table), then you don't need to use the VFPTABLE function. If, however, you would like to run the program in a batch, unattended mode (for example, if you were looping through a list of .DBF files in a directory and calling ADD2DBC.PRG for each .DBF file), use a program like ADD2DBC.PRG:

```
* Program Name: ADD2DBC.PRG
*
* Auto-answer the 'Table Update Warning' message
* that comes up when you add a FoxPro 2.x-style
* table to a VFP 3.0 database:
*
* File '<dbf_name>.dbf' will be updated to a new
* file format, making it unreadable to previous
* versions of FoxPro. This can be reversed with
* the 'COPY TO' command, and the old table will
* be saved as a .BAK file. Continue?
*
*                    <<Yes>>            <No>
*                      -                 -
PARAMETERS tcFileName

IF NOT VfpTable(tcFileName)
  * Auto-answer the 'Table Update Warning' message
  KEYBOARD 'Y' PLAIN
ENDIF
ADD TABLE (tcFileName)
* Auto-erase the .BAK file that was created
ERASE  (tcFileName + '.BAK')
```

As you deal with the complications brought on by having to deal with two different types of files2.x and 3.0 tables -- that have the same extension, it's useful to have tools on hand that make short work of some of the problems you run into. This technique will help with a couple of those problems.