February, 2002

Advisor Answers

Retrieving a file's creation date

VFP 7/6/5/3

Q: How can I get the creation date and time of a file in FoxPro?

–Dr. Deniz Yüksel (via Advisor.COM)

A: FoxPro provides several native ways to get the most recently modified date of a file (check out ADIR(), FDATE() and FTIME()), but you have to go outside the product to get the creation date.

There are also a couple of ways to get the creation date of a file. One of them is much easier than the other, so I'll look at that approach, which uses the Windows Scripting Host's FileSystemObject, first.

The Windows Scripting Host (WSH) is an Automation wrapper for a tremendous number of system operations. It provides an easy way to do a lot of things that used to require API calls. You'll find a good overview of the Windows Scripting Host in Gene Sally's articles in the May '99 and August '99 issues of FoxPro Advisor.

The object we're interested in is the FileSystemObject. You can create an instance like this:

```
oFSO = CreateObject( "Scripting.FileSystemObject" )
```

This object lets you work with drives, folders, and files. The object has a GetFile method that gives you an object reference to a particular file. For example, to get a reference to Browser.APP, use code like this:

```
oFT = oFSO.GetFile( HOME() + "Browser.APP" )
```

Once you have a reference to a file, you can find pretty much anything you'd want to know about it. The DateCreated property contains the information you're looking for:

```
tCreated = oFT.DateCreated
```

The file object also has DateLastAccessed and DateLastModified properties, among others.

With such an easy solution, why even look at another approach? Because not every machine has the WSH installed. Some people don't have it because they're using older versions of Windows and Internet Explorer that didn't include it. Others don't have it because some companies have decided that the risks of this tool outweigh its benefits. The WSH can be used for all kinds of anti-social behavior (think viruses).

So, a look at the more complex solution is in order. This version uses API functions, including a couple that require structures. While the structures involved are quite simple and we could create them by hand, the code is easier to write and easier to read if we use the Struct class created and placed in the public domain by my co-columnist, Christof. The Struct class lets you work with VFP objects rather than structures and handles all the details of converting the objects into the form you need to pass to API functions, and converting the returned data back into the VFP objects.

The first step in getting the file creation date is to get a handle to the file. That's because the API function that returns file dates doesn't accept the file name as a parameter; it requires a handle.

The OpenFile function opens a file for all kinds of uses and returns a handle to the file. (VFP's low-level file functions use handles in the same way.) As with all API functions, you have to declare OpenFile before using it.

```
DECLARE INTEGER OpenFile IN WIN32API ;
   STRING lpFileName, STRING lpReOpenBuff, LONG wStyle
```

The first parameter to OpenFile is the filename, including the path. The second parameter returns information you'd use to reopen the file with a later call to OpenFile, and can be safely ignored for our purposes – just pass a long, empty string. The last parameter indicates the purpose for opening the file – pass 0 to indicate the file is open for reading.

This call opens Browser.APP. (Keep in mind that API functions are case-sensitive.)

```
nHandle = OpenFile(HOME() + "Browser.APP", SPACE(128), 0)
```

The next step is to retrieve the dates for the file. The GetFileTime API function gives you access to the creation, last accessed and last modified dates. Each of those is placed into a FileTime structure.

This is where Christof's Struct class starts to come in handy. To use it, you need to point to the necessary class libraries, then create the appropriate object. We'll need a FileTime object for each of the dates the function returns:

```
SET CLASSLIB TO Struct, WinStruct
oCreated = CreateObject("FileTime")
oAccessed = CreateObject("FileTime")
oModified = CreateObject("FileTime")
```

In most cases, we can substitute a character string for a structure. Here's the declaration for the GetFileTime function:

```
DECLARE LONG GetFileTime IN WIN32API ;
   LONG hFile, STRING @ lpCreationTime, ;
   STRING @ lpLastAccessTime, STRING @ lpLastWriteTime
```

To pass our structures to the function, we need to convert them to strings. The GetString method of the structure class does this:

```
cCreated = oCreated.GetString()
cAccessed = oAccessed.GetString()
cModified = oModified.GetString()
```

At this point, we can call the function, passing the strings by reference:

```
nError = GetFileTime( nHandle, @cCreated, ;
                      @cAccessed, @cModified )
```

The function returns 0 if an error occurs and a non-zero value if it's successful. Once the function returns, we need to store the updated dates back into the structure objects. The structure class's SetString method does the trick.

```
oCreated.SetString( cCreated )
oAccessed.SetString( cAccessed )
oModified.SetString( cModified )
```

Because Windows uses UTC (universal coordinated time) internally, we need to convert from that time to the local time. The FileTimeToLocalTime function handles this chore. It uses another FileTime structure. Here's the declaration for that function:

```
DECLARE INTEGER FileTimeToLocalFileTime in Win32API ;
   STRING lpFileTime, STRING @ lpLocalFileTime
```

This code sets up and performs the function call:

```
oLocalTime = CREATEOBJECT("FileTime")
cLocalTime = oLocalTime.GetString()
```

```
FileTimeToLocalFileTime( oCreated.GetString(), ;
                         @cLocalTime)
```

The final step is to convert the date information into a readable format. The API function for that is FileTimeToSystemTime. The function takes two parameters, the FileTime structure you want to convert and a SystemTime structure to contain the result. Here's the declaration, again substituting strings for structures:

```
DECLARE LONG FileTimeToSystemTime IN WIN32API ;
  STRING lpFileTime, STRING @ lpSystemTime
```

As before, we need to create the structure, using the right class, and convert it to a string:

```
oCreatedTime = CreateObject( "SystemTime" )
cCreatedTime = oCreatedTime.GetString()
```

To call the function, pass the string version of the created time and the new system time string

```
FileTimeToSystemTime( cLocalTime, @cCreatedTime )
```

As with GetFileTime and FileTimeToLocalTime, a non-zero return value indicates success. In that case, the oCreatedTime object has properties that represent the different components of the date. You can assemble them using DATETIME(), like this:

```
WITH oCreatedTime
   tResult = DATETIME( .wYear, .wMonth, .wDay, ;
                       .wHour, .wMinute, .wSecond )
ENDWITH
```

This month's Professional Resource CD contains FileCreated.PRG, a function that accepts a filename and returns its creation date, using this approach, as well as Struct.ZIP, the source code and documentation for Christof's structure classes.

–Tamar