Visual Studio Magazine Online

Classic VB Corner

Lemme Tell Ya Where To Stick It

ONLINE ONLY

In these days of increasingly common least-privileged users, where should you store your application data?

January 19, 2009 · by Karl E. Peterson

In my last column, I provided code and encouragement to use INI files when appropriate to store your application settings. Back in the proverbial Good Olde Days, you could simply store your INI files wherever your EXE happened to find itself. What could be easier? Finding your INI file was as quick and easy as

```
App.Path & "\" & App.EXEName & ".ini"
```

Of course, you still had to guard against the occasional user who'd stick your EXE in the root folder of a drive, but that's about as difficult as it got. Now, not only do ordinary users lack write permissions to the Program Files folder, but there are more and more folks taking advantage of the roaming support offered by today's networks. Roaming users can sit down at any machine on the network, log in and expect that all their settings will be the way they left them on the last machine they logged off from. The concepts of roaming users and least privilege have made the developer's life more difficult.

Microsoft offers scads of recommendations or "Best Practices" on where to put your application data, based in part on how the user will be using it. These have been "in effect" for years, of course, but it's only recently that they've been enforced.. The key points to pull are that application data generally breaks down into three categories: that used by all users of the application, that used by a specific user, and that used by specific users when roaming about the network. Windows, conveniently, offers three predefined folder locations for storing these three distinct kinds of data.

To find the locations of these and other special system folders, you will generally want to call the SHGetFolderPath API function. However, this function is exported from shfolder.dll which wasn't introduced until Internet Explorer 5, so that presents a bit of an issue for older systems. While many Win9x systems will have IE 5 installed, you still need to be prepared to fall back to older APIs for system folders. Unfortunately, those older shells don't offer this same concept of three distinct application data classes, so you're faced with a dilemma.

You can call SHGetFolderPath, if it's available, and use the path presented for your needs. Or, on Win9x systems, you can simply punt and stick with the App.Path for application data. I honestly recommend the latter approach, after much testing, because the various scenarios raised by the potential lack of shfolder.dll are just too fraught with peril to be worth the effort. And the number of robust multi-user Win9x systems (say that without laughing!) has to be very small indeed.

That leaves us with NT-class systems, all of which (back to NT4, if IE 5 is installed) fully support the three distinct data classes. SHGetFolderPath accepts a wide array of constants to request specific folder locations. The three we're interested in are CSIDL_APPDATA, CSIDL_LOCAL_APPDATA and CSIDL_COMMON_APPDATA. Typically, based on operating system, they'll point to the following locations:

CSIDL_APPDATA	
Windows NT4	C:\WINNT\Profiles\%USERNAME%\Application Data
Windows 2000/XP	C:\Documents and Settings\%USERNAME%\Application Data
Windows Vista	C:\Users\%USERNAME%\AppData\Roaming

CSIDL_LOCAL_APPDATA	
Windows NT4	C:\WINNT\Profiles\%USERNAME%\ Local Settings\Application Data
Windows 2000/XP	C:\Documents and Settings\%USERNAME%\ Local Settings\Application Data
Windows Vista	C:\Users\%USERNAME%\AppData\Local

CSIDL_COMMON_APPDATA	
Windows NT4	C:\WINNT\Profiles\All Users\Application Data
Windows 2000/XP	C:\Documents and Settings\All Users\Application Data
Windows Vista	C:\ProgramData

With all three folder types, the general idea is to create subfolders below them for your data. You'd first create a folder with your company name, then a subfolder below that with the product name. If your data is version-specific, you can create a further subfolder for each subsequent version.

Let's look at each of them, in reverse order. The CSIDL_COMMON_APPDATA folder is where you'll want to store application data intended for all users to share. This is analogous to storing data in the HKEY_LOCAL_MACHINE hive of the registry. There's a good chance this will be a write-once opportunity, to be performed at install time, as that execution will almost certainly be done with administrative privileges. Ordinary users will most likely not have write access to this folder.

The CSIDL_LOCAL_APPDATA folder is used for application data that will always be local to the current machine, but is set aside on a per user basis. The data in this folder is not available on a roaming basis, so it should be data that the user will likely not miss if they log in to a different machine. The best example of this sort of application data are the temporary files stored by Internet browsers. It would consume far too much time and bandwidth to sling these back and forth to the server at every log-in and log-out.

Finally, the CSIDL_APPDATA folder is the one you will likely be most interested in. Data stored here is available to roaming users at whatever machine they log in to. This is the best place to store simple configuration data. All users have write access to this (and the last) folder. Note that none of the above folders are for user-generated data! That would properly belong under the My Documents hierarchy.

So, with those preliminaries out of the way, I'll just point you to what I think many of you read this column for (grin). I've written a drop-in ready class that you can use in any 32-bit version of Classic VB or VBA to uncover where the system is hiding all these special folders. Hit my Web site and download the SysFolders sample as soon as you're done reading here. It'll be a natural companion to the kpINI sample I wrote about last time. The CSystemFolders class offers an enumeration of all the CSIDL constants, so you can retrieve (and even force the creation of) any of the special folders. To make it complete, it also offers simple wrappers for the GetWindowsDirectory, GetSystemDirectory and GetTempPath API calls.

In cases where SHGetFolderPath isn't available, CSystemFolders reverts to doing the best it can, by calling the SHGetPathFromIDList and SHGetSpecialFolderLocation APIs instead. Of course, the list of folders these APIs know about is far shorter. Generally, you'll only run into this situation on older Windows 95 and 98 systems that haven't even been updated to include IE 5. The class offers an Enhanced property so you can immediately know whether the shell has been updated to include this functionality.

Finally, a word of caution: In testing for this column, I ran into a very puzzling situation. I found that SHGetFolderPath was working in a "clean" Windows 95 virtual machine that only had IE 3! This confounded me for a while. Finally, it hit me -- the "Virtual Machine Additions" offered by VirtualPC 2004 installed the shell update. This was, of course, the only thing I'd installed to what I thought was a "clean" system. Ouch. That's what a guy gets for trying to make the testbed as comfortable as possible, eh?

About the Author

Karl E. Peterson wrote Q&A, Programming Techniques, and various other columns for VBPJ and VSM from 1995 onward, until Classic VB columns were dropped entirely in favor of other languages. Similarly, Karl was a Microsoft BASIC MVP from 1994 through 2005, until such community contributions were no longer deemed valuable. He is the author of VisualStudioMagazine.com's new Classic VB Corner column. You can contact him through his Web site if you'd like to suggest future topics for this column.