When will GetSystemWindowsDirectory return something different from GetWindowsDirectory?

devblogs.microsoft.com/oldnewthing/20140723-00

July 23, 2014

Raymond Chen

Most of the time, the `GetWindowDirectory` returns the Windows directory. However, as noted in the documentation for `GetSystemWindowsDirectory`:

> With Terminal Services, the **GetSystemWindowsDirectory** function retrieves the path of the system Windows directory, while the **GetWindowsDirectory** function retrieves the path of a Windows directory that is private for each user. On a single-user system, **GetSystemWindows-Directory** is the same as **GetWindowsDirectory**.

What's going on here, and how do I test this scenario? When Terminal Services support was being added to Windows NT 4.0 in the mid 1990's, the Terminal Services team discovered that a lot of applications assumed that the computer was used by only one person, and that that person was a local administrator. This was the most common system configuration at the time, so a lot of applications simply assumed that it was the *only* system configuration. On the other hand, a Terminal Server machine can have a large number of users, including multiple users connected simultaneously, and if the Terminal Services team took no special action, you would have found that most applications didn't work. The situation "most applications didn't work" tends not to bode well for adoption of your technology. Their solution was to create a whole bunch of compatibility behaviors and disable them if the application says, "Hey, I understand that Terminal Server machines are different from your average consumer machine, and I know what I'm doing." One of those compatibility behaviors is to make the `GetWindowsDirectory` function return a private writable directory rather than the real Windows directory, because old applications assumed that the Windows directory was writable, and they often dumped their private configuration data there. The signal to disable compatibility behaviors is the `IMAGE_DLLCHARACTER-ISTICS_TERMINAL_SERVER_AWARE` flag in the image attributes of the primary executable. You tell the linker that you want this flag to be set by passing the `/TSAWARE:YES` parameter on the command line. (At some point, the Visual Studio folks made `/TSAWARE:YES` the default for all new projects, so you are probably getting this flag set on your files without even realizing it. You can force it off by going to Configuration Properties, and under Linker, then System, change the "Terminal Server" setting to "Not Terminal Server Aware".) Note that

only the flag state on the primary executable has any effect. Setting the flag on a DLL has no effect. (This adds to the collection of flags that are meaningful only on the primary executable.)

The other tricky part is that the Terminal Server compatibility behaviors kick in only on a Terminal Server machine. The way you create a Terminal Server machine has changed a lot over the years, as has the name of the feature.

- In Windows NT 4.0, it was a special edition of Windows, known as Windows NT 4.0 Terminal Server Edition.
- In Windows 2000, the feature changed its name from Terminal Server to Terminal Services and became an optional server component rather than a separate product. You add the component from Add/Remove Programs.
- In Windows Server 2003 and Windows Server 2008, you go to the Configure Your Server Wizard and add the server rôle "Terminal Server."
- In Windows Server 2008 R2, the feature changed its name again. The instructions are the same as in Windows Server 2008, but the rôle name changed to "Remote Desktop Services".
- In Windows Server 2012, the feature retained its name but became grouped under the category "Virtual Desktop Infrastructure." This time, you have to enable the rôle server "Remote Desktop (RD) Session Host."

Terminal Services is the Puff Daddy of Windows technologies. It changes its name every few years, and you wonder what it will come up with next.

Raymond Chen

Follow