Beating The System: Windows 2000, Are You Getting Certified?

by Dave Jewell

As promised last time, we're taking a break from the development of our file system component to concentrate on some of the more important programming issues surrounding Windows 2000, with a special emphasis on what you need do in order to be logo compliant. As was the case with Windows 95 and 98 (and even Windows 3.1, for that matter), Microsoft have released a set of logo requirements prior to the roll-out of the operating system itself. If you want to display the coveted *'Certified for Windows'* gold logo on your product's box, then you need to be logo compliant. To me, this is highly reminiscent of those occasions when my eight-year-old son arrives home proudly sporting a gold star from his teacher, and many developers feel that they've got better things to do with their time than pander to Microsoft's demands.

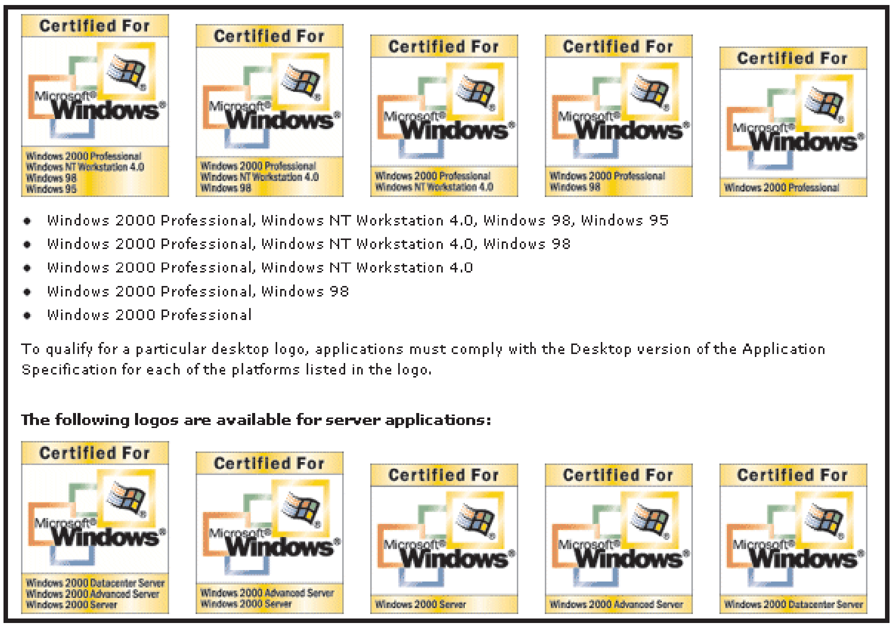
The official reason for having a set of logo requirements is that it improves standardisation and thereby gives end-users a common experience when working with any logo compliant product. This is undeniably true: any initiative which gives people a smooth, consistent experience is bound to be a good thing, both in terms of customer satisfaction and in terms of reduced support overhead.

However, there are those who believe that Microsoft's logo requirements are also self-serving, for at least two reasons. Firstly, they lock developers into new, proprietary Microsoft technologies which are only available on the Windows platform, and secondly, they divert third-party development effort away from the creation of innovative new applications, forcing small developers to concentrate on peripheral issues which don't always relate directly to the functionality of the software, and thereby increasing the competitiveness of Microsoft's own applications.

A past example of this was the 16-bit versus 32-bit code issue. Back in the days of Windows 95, at a time when 32-bit programming was relatively new, the logo requirement stipulated that, in order to be compliant, an application had to be fully implemented using 32-bit code. From the end-user's perspective, there was no clear benefit to be gained by imposing such a restriction, and this requirement rather ignored the fact that much of the code in Windows 95 (and today, Windows 98) is still implemented in 16-bit DLLs. It was really a case of 'do what we tell you, even though we don't do it ourselves'.

Be that as it may, the remainder of this article concentrates on some of the key requirements introduced by the Windows 2000 logo programme. As you'll see from Figure 1, there are various 'levels' of logo certification, the 'deluxe' version being reserved for applications that have been tested for compliance under Windows 2000 Professional, NT 4.0 Workstation, Windows 98 and Windows 95. If you want to read the logo requirements for yourself, the first place to check is http://msdn.microsoft.com/winlogo where you'll find links to other relevant documentation and downloadable information. In particular, if you point your web browser at http://msdn.microsoft.com/certification/appspec.asp, you'll find a downloadable Word document

➤ *Figure 1: The coveted gold stars... err... I mean logos. If you really don't give a toss about getting Microsoft's seal of approval, then you'll undoubtedly have less work to do in getting things working under Windows 2000, but depending on what corners you cut, you may get some flack from your customers.*


Certified For Microsoft Windows
Windows 2000 Professional
Windows NT Workstation 4.0
Windows 98
Windows 95
Windows 2000 Datacenter Server
Windows 2000 Advanced Server
Windows 2000 Server

- Windows 2000 Professional, Windows NT Workstation 4.0, Windows 98, Windows 95
- Windows 2000 Professional, Windows NT Workstation 4.0, Windows 98
- Windows 2000 Professional, Windows NT Workstation 4.0
- Windows 2000 Professional, Windows 98
- Windows 2000 Professional

To qualify for a particular desktop logo, applications must comply with the Desktop version of the Application Specification for each of the platforms listed in the logo.

The following logos are available for server applications:

➤ *Facing page: Table 1*

W95	W98	W2000	Feature
Fundamentals			
Yes	Yes	Yes	Perform primary functionality and maintain stability.
Yes	Yes	Yes	Provide 32-bit components.
Yes	Yes	Yes	Support Long File Names and UNC paths.
Yes	Yes	Yes	Support printers with long names and UNC paths.
No	No	Yes	Do not read from or write to Win.ini, System.ini, Autoexec.bat or Config.sys on any Windows operating system based on NT technology.
Yes	Yes	Yes	Ensure non-hidden files have associated file-types, and all file-types have associated icons, descriptions, and actions.
Yes	Yes	Yes	Perform Windows version checking correctly.
Yes	Yes	Yes	Support AutoPlay of compact discs.
No	No	Yes	Kernel mode drivers must pass verification testing on Windows 2000.
No	Yes	Yes	Hardware drivers must pass WHQL testing.
Install/Uninstall			
Yes	Yes	Yes	Install using a Windows Installer-based package that passes validation testing.
Yes	Yes	Yes	Observe rules in componentisation.
Yes	Yes	Yes	Identify shared components.
Yes	Yes	Yes	Install to Program Files by default.
Yes	Yes	Yes	Support Add/Remove Programs properly.
Yes	Yes	Yes	Ensure that your application supports advertising.
Yes	Yes	Yes	Ensure correct uninstall support.
Component Sharing			
No	No	Yes	Do not attempt to replace files that are protected by System File Protection.
No	Yes	Yes	Component producers: Build side-by-side components.
No	Yes	Yes	Application developers: Consume and install side-by-side components.
Yes	Yes	Yes	Install any non side-by-side shared files to the correct locations.
Data and Settings Management			
Yes	Yes	Yes	Default to My Documents for storage of user-created data.
Yes	Yes	Yes	Classify and store application data correctly.
No	No	Yes	Degrade gracefully on access denied.
No	No	Yes	Run in a secure Windows environment.
No	No	Yes	Adhere to system-level Group Policy settings.
User Interface Fundamentals			
Yes	Yes	Yes	Support standard system size, colour, font and input settings.
Yes	Yes	Yes	Ensure compatibility with the High Contrast option.
Yes	Yes	Yes	Provide documented keyboard access to all features.
Yes	Yes	Yes	Expose the location of the keyboard focus.
Yes	Yes	Yes	Do not rely exclusively on sound.
Yes	Yes	Yes	Do not place shortcuts to documents, help, or uninstall in the Start Menu.
No	Yes	Yes	Support multiple monitors.
OnNow/ACPI Support			
No	Yes	Yes	Indicate busy application status properly.
No	Yes	Yes	Respond to sleep requests from the operating system properly.
No	Yes	Yes	Handle sleep notifications properly.
No	Yes	Yes	Handle wake from normal sleep without losing data.
No	Yes	Yes	Handle wake from critical sleep properly.
Application Migration			
Yes	Yes	No	Application must continue to function after upgrade to Windows 2000 Professional without reinstall.

Design Guide for Building Business Applications which details the requirements for ordinary (client only) applications. There are also other documents for server-only and client/server applications. Strictly speaking, these requirements documents are subject to change, but let's not bite our fingernails too much, worrying at the prospect of last minute changes to the specification.

The most important logo requirements are summarised in Table 1, which I've taken from Microsoft's preliminary documentation. As you can see, it compares the various logo requirements for each of Windows 95, 98 and Windows 2000. There was an additional column for NT 4, but I removed this for the sake of brevity.

Installation Changes: The Joy Of MSI

If you're a reader of *Developers Review*, you'll know that I've recently reviewed the Wise for Windows Installer package, which is one of a new breed of software installers designed to work with Microsoft's new MSI technology. MSI technology (also known, somewhat confusingly, as 'Windows Installer') simply means that Microsoft have effectively taken control of the software installation process. Rather than allowing third-party software installers to load applications onto a Windows 2000 system, Microsoft think that they can do the job better themselves.

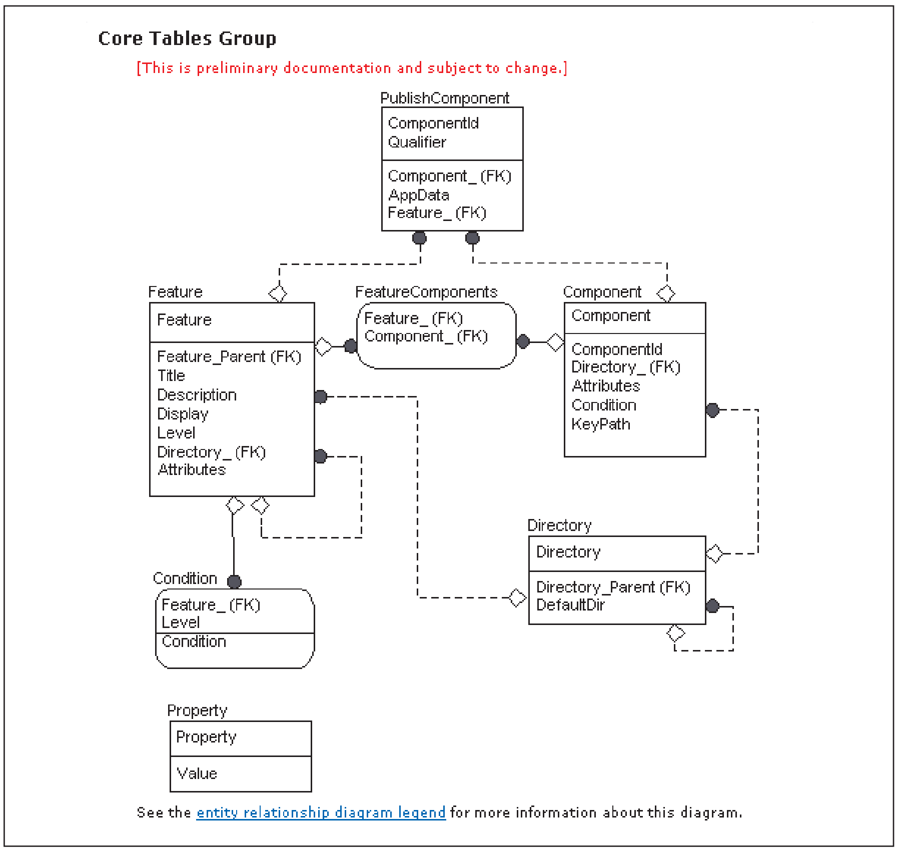
Arguably, Microsoft's track record, in terms of software reliability, isn't any better than anyone else's (keep quiet at the back, there!), but it does make sense to have the operating system responsible for application installation. By doing things this way, the software installer can better integrate with new operating system gizmos such as the enhanced Add/Remove Programs database and can incorporate advanced capabilities such as the ability to 'roll-back' a failed install.

Under Windows 2000, a third-party software setup is packaged as a specialised type of file with an extension of .MSI. This file type is a sort of encrypted, compressed database and things are arranged such that you need only double-click an MSI file in order to launch the new Microsoft installer.

If you want to earn Microsoft's much-coveted Windows 2000 compatibility logo for your product, then you *have* to ship your product with an MSI compatible setup. This presents an obvious problem when installing on (say) Windows 98 and Windows 95 but, interestingly, it's possible to 'retro-fit' the installer engine onto both these platforms. Thus, the cunning Wise for Windows Installer product will let you create a logo-compatible setup which works as advertised for Windows 2000. If the setup program detects that it's not running on W2000, it installs the Microsoft Installer engine first, and then transfers control to the MSI file. This ability to retro-fit the MSI engine onto Windows 95 and Windows 98 is reflected in Table 1, where you'll see a 'Yes' against all three operating systems for the 'Install using a Windows Installer-based package...' entry.

Speaking personally, I think the need for a consistent software installation procedure is one of the logo requirements that I feel most enthusiastic about. Just the other day, I installed a desktop enhancement utility called Nuts n' Bolts. This was supplied on the cover CD of a UK computer magazine which had better remain nameless: I write for them too and they might not appreciate the adverse publicity! Suffice it to say that not long after I'd installed Nuts n' Bolts, I found that my Windows 98 setup was behaving very oddly. I could not load Microsoft Developer Studio, Visual Basic 6.0 was falling over and Microsoft's Deluxe CD player crashed every time I put an audio CD in the drive. It turned out that Nuts n' Bolts had installed

➤ *Figure 2: So you thought software installers were simple? This is the entity relationship diagram which describes the various tables found within an MSI file. It's essentially a relational database which describes the various components present within a particular software installation.*



antique copies of OLEAUT32.DLL, MSVCRT.DLL and a couple of other DLLs, the result being that my PC was very unhappy indeed. There's a lot more I could say about Nuts n' Bolts, but one word will suffice: *avoid*.

The important point, of course, is that a setup program should *never* replace system DLLs with older versions. Moreover, it should check disk space requirements before installation, increment the reference count of shared DLLs, and so forth. By using the new Microsoft installer technology, most of these issues are taken care of for you.

Time To Emigrate?

If you're involved in creating a new application now, it would also be prudent to consider migration issues. Migration is essentially Microsoft-speak for the process of moving an existing Windows 95/98 setup to Windows 2000. Ideally, when a computer system is migrated to Windows 2000, existing applications will continue to work as before. Ideally! In practice, I still have a PC containing a 56K WinModem which disappeared from the face of the planet on the day that I upgraded from Windows 95 to Windows 98. No matter what patches I download and install, that modem is still non-existent as far as Windows is concerned, despite the fact that the card is still installed in the machine. I find it very hard to believe that the card developed a hardware fault on the very day that I installed Windows 98. Rather, this is an example of a migration that didn't work out, and I've heard numerous examples from other users.

The fact is, an application that ran perfectly well under Windows 9x will occasionally fail when it wakes up and finds itself running under Windows 2000. As a responsible software developer, you should do everything possible to preclude the end-user having to reinstall your product from scratch. One of the main reasons why a program might not work under Windows 2000 relates to the changed layout of certain system registry entries, or there might be other problems relating to changes in system DLLs. Fortunately, Microsoft have come to the rescue here courtesy of a new feature called MEI or the Migration Extension Interface. Basically, you create a custom DLL and deploy this along with your application. During system migration, the Windows 2000 setup program will automatically call your migration DLL, thus giving you control at certain times during the migration process. You can then perform whatever application-specific actions are needed to migrate your program to Windows 2000, such as moving system registry entries, enabling other platform-specific features, and so forth.

The migration DLL that you supply is just an ordinary DLL and must export a number of named routines (seven in all) that are called at various times during the Windows 2000 setup process. If your DLL doesn't contain all seven of these routines, then it won't be recognised as a valid migration DLL and will be ignored. When you first install your application, you must create a registry entry that points to your migration DLL. This must be located under the following registry path:

```
HKEY_LOCAL_MACHINE\Software\
  Microsoft\Windows\
  CurrentVersion\Setup\
  Migration DLLs
```

There's no guarantee that your migration DLL will be called in any specific order (relative to other migration DLLs) and you need to bear this in mind if you're developing a suite of applications, each of which has its own migration DLL. For more information on migration DLLs, and the seven functions that you need to export, you can check out the appropriate Microsoft documentation at

```
http://msdn.microsoft.com/
  library/psdk/win9xmig/
   migext_2jvp.htm
```

Still on the subject of installation and setup issues, you should be aware that Windows 2000 supports 'side-by-side sharing', which is a fancy way of saying that different processes can load different versions of the same DLL, whether it be a plain-vanilla DLL or a COM server, such as an ActiveX control.

You may remember the debacle which ensued when Microsoft released a beta version of Internet Explorer 4.0. Everything worked fine apart from the fact that the installation installed a buggy version of COMCTL32.DLL, the common controls library. This DLL worked fine with IE4.0, but didn't seem to get on well with any other application.

I wholeheartedly approve of Microsoft releasing beta versions of their products, but not if doing so is going to screw up an existing Windows installation! The simplest way of avoiding the problem would have been for Microsoft to install the new COMCTL32 library into the *same directory* as the other Internet Explorer components that needed it, leaving the 'global' version of the common controls library alone. That, in a nutshell, is the essence of 'side-by-side sharing'.

In early versions of Windows, it was impossible for two different versions of the same DLL (or EXE file, for that matter) to be in memory at the same time. Note carefully that I am not talking about different instances of a running program or DLL here: in these cases, there is only ever one copy of the code in memory. Rather, I'm saying that in early Windows implementations, you could not have two DLLs in memory, both of which had the same module name: the operating system would not allow it. This restriction no longer exists in either Windows 98 or Windows 2000. You simply need to ensure that special versions of otherwise standard DLLs reside in the same directory as the application that needs them. In the case of side-by-side COM components, the 'special' version should be registered using a relative (rather than absolute) pathname, and should use a different GUID to the standard issue.

System File Protection

Another component sharing issue relates to the new SFP (System File Protection) feature which is now a part of Windows 2000. Basically, there's now a little process which lurks in the background checking to see whether some badly behaved application has accidentally (or deliberately!) deleted or overwritten one of the operating system's critically important system files. If such a misdemeanour is detected the file in question is immediately restored from a backup.

At the risk of causing offence, I have to say that this strikes me as a somewhat 'cloth-cap' approach to protecting the integrity of the core operating system. Surely, with all the emphasis on security that NT now has, it would have made more sense to trap access to critical files at the file system level and simply deny the attempted write, rename, deletion or whatever. I feel that this approach would have made for a much more secure system. After all, what's to stop the adventurous hacker from finding some way to dispose of the aforementioned background task, or even massage the backup copies of the system DLLs before initiating a restore operation? Time will tell, but I would have liked to see a much more serious approach to the problem of protecting operating system integrity. It goes without saying, incidentally, that only authorised Microsoft service packs and operating system upgrades are allowed to make changes to protected files, but I don't doubt that someone will work around this, given time.

To enumerate the list of protected system files, you can use the `SfcGetNextProtectedFile` API routine which is prototyped as follows:

```
BOOL SfcGetNextProtectedFile(
  PPROTECTED_FILE_DATA
  ProtFileData);
```

The `PROTECTED_FILE_DATA` data structure is formatted as shown in Listing 1.

For each filename retrieved, the `FileName` field is filled with the name of the file. You should set `FileNumber` to zero before calling the function for the first time. The routine will return non-zero on

Data Type	Folder CSIDL	Folder Location
Per user, roaming	CSIDL_APPDATA	[user profile]\Application data
Per user, non-roaming	CSIDL_LOCAL_APPDATA	[user profile]\Local Settings\ Application data
Per machine (non-user specific and non- roaming)	CSIDL_COMMON_APPDATA	All Users\Application data

➤ *Table 2*

```
typedef struct _PROTECTED_FILE_DATA {
    WCHAR   FileName[MAX_PATH];
    DWORD   FileNumber;
} PROTECTED_FILE_DATA, *PPROTECTED_FILE_DATA;
```

➤ *Listing 1*

success and zero when there are no more files to enumerate.

There's also another routine, `SfcIsFileProtected`, which can be used to determine whether an arbitrary filename is protected. It is prototyped:

```
BOOL SfcIsFileProtected(
  LPCWSTR ProtFileName);
```

Nothing New Under The Sun?

One of the most interesting changes in Windows 2000 is the deprecating of the system registry as the best place to store application data. Let's briefly recap: when Windows first crawled out of the primordial ooze, all we had to work with were a couple of .INI files called WIN.INI and SYSTEM.INI. Pretty soon, every man and his dog started writing shed-loads of application-specific data into these all-important files. In a panic (remember folks, the implementation limited these files to 64Kb in size!) Microsoft came up with the idea of private .INI files, meaning that developers could now store their configuration data into a separate file of their own choosing, preferably in the same directory as the program.

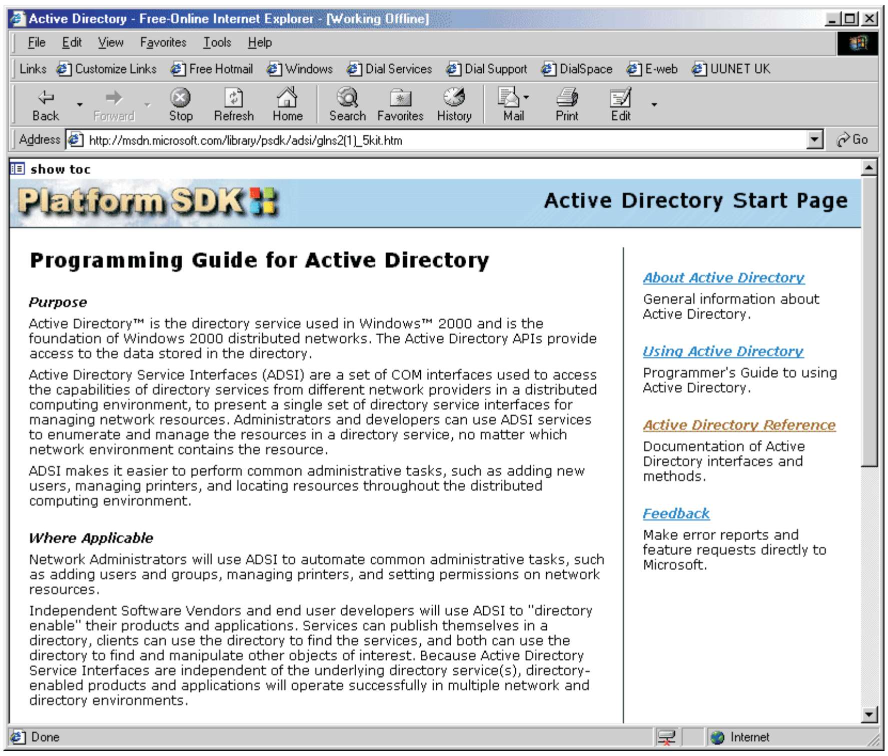
Time moved on, and some deep thinker at Redmond hit upon the idea of a single, central repository for everything: operating system data *and* application information. Thus the concept of the system registry was born. Now, a few years on, we have a situation where it's routine to have vast system registry files (the one on my primary development machine is over 8Mb in size) and any damage to this single file can quickly reduce a PC to the high-tech equivalent of a chocolate teapot. Now, things have come full circle once more and with the advent of Windows 2000, we're encouraged to save our data into special directories reserved for the purpose.

Actually, I'm not being entirely fair here. What we're really talking about is Microsoft's new 'Data and Settings Management', which embraces not only application configuration settings but document files as well. Windows 2000 provides support for 'roaming users', which basically means that, in a networked environment, it should be possible for a user to move between different workstations and find that user settings and configuration data transparently move with him or her.

This is accomplished by having the application write user files to a special folder which has been designated by the system administrator. Under Windows 2000, the `SHGetFolderPath` API call has been extended to return more folder locations than previously. Thus, passing `CSIDL_PERSONAL` to `SHGetFolderPath` will retrieve the pathname of that user's My Documents folder. In a networked environment, this folder can be located on a network and thus 'follow the user around'. In fact, if you call the common dialogs' file open and save routines with no pathname specified, they will always default to the My Documents location. Microsoft recommend that you place other folders below the My Documents folder including My Pictures for use by graphics applications. Obviously, there's nothing to stop punters from storing files anywhere they want, but Microsoft encourage you to gently point end-users at the My Documents directory tree by default.

As regards configuration data (as opposed to documents) Microsoft now make it plain that the registry isn't the best place to store large amounts of configuration information. They define 'large amounts' as anything larger than 64Kb in size but, personally, I would be very unhappy to store anything like this amount of data in the registry. In addition to the aforementioned `CSIDL_PERSONAL`

➤ *Figure 3: If you don't have an MSDN subscription, and you don't feel like downloading the 650Mb (yes, really!) SDK from the Microsoft site, you can nevertheless browse the MSDN library online. Here's the information that relates to the new ADSI (Active Directory Service Interfaces).*


Active Directory - Free-Online Internet Explorer - [Working Offline]
File Edit View Favorites Tools Help
Links Customize Links Free Hotmail Windows Dial Services Dial Support DialSpace E-web UUNET UK
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address http://msdn.microsoft.com/library/psdk/adsi/glns2(1)_5kit.htm Go
show toc
Platform SDK
Active Directory Start Page
Done
Internet

Programming Guide for Active Directory

Purpose

Active Directory™ is the directory service used in Windows™ 2000 and is the foundation of Windows 2000 distributed networks. The Active Directory APIs provide access to the data stored in the directory.

Active Directory Service Interfaces (ADSI) are a set of COM interfaces used to access the capabilities of directory services from different network providers in a distributed computing environment, to present a single set of directory service interfaces for managing network resources. Administrators and developers can use ADSI services to enumerate and manage the resources in a directory service, no matter which network environment contains the resource.

ADSI makes it easier to perform common administrative tasks, such as adding new users, managing printers, and locating resources throughout the distributed computing environment.

Where Applicable

Network Administrators will use ADSI to automate common administrative tasks, such as adding users and groups, managing printers, and setting permissions on network resources.

Independent Software Vendors and end user developers will use ADSI to "directory enable" their products and applications. Services can publish themselves in a directory, clients can use the directory to find the services, and both can use the directory to find and manipulate other objects of interest. Because Active Directory Service Interfaces are independent of the underlying directory service(s), directory-enabled products and applications will operate successfully in multiple network and directory environments.

About Active Directory

General information about Active Directory.

Using Active Directory

Programmer's Guide to using Active Directory.

Active Directory Reference

Documentation of Active Directory interfaces and methods.

Feedback

Make error reports and feature requests directly to Microsoft.

parameter, SHGetFolderPath now takes a number of additional parameters as shown in Table 2.

CSIDL_APPDATA refers to application data which contains user-specific configuration information that's not tied to a particular machine. This might, for example, map to the following path:

```
C:\Documents and Settings\
  Dave Jewell\Application Data
```

The CSIDL_LOCAL_APPDATA specifier is used to denote a directory sub-tree which, again, contains user-specific data but which, this time, is also specific to the current machine. Conceptually, this is still part of the user's profile (and is therefore inaccessible to other users). A typical example might be:

```
C:\Documents and Settings\
  Dave Jewell\Local Settings\
  Application Data
```

Finally, CSIDL_COMMON_APPDATA designates configuration information which is applicable to all users of this computer and is therefore, by definition, non-roaming. Once again, a typical path might be:

```
C:\Documents and Settings\
  All Users\Application Data
```

You should bear in mind that SHGetFolderPath may not be present on 'lesser' platforms. SHGetFolderPath is implemented inside a small DLL called SHFOLDER.DLL and ideally you should place calls to SHGetFolderPath inside a wrapper which checks for the presence of the DLL or, better yet, redistribute the DLL as part of your application: Microsoft encourage you to do this. The DLL was shipped with NT Service Pack 4, Windows 98 Second Edition, and I believe that it will also be present whenever Internet Explorer 5 has been installed.

OnNow And Off Now!

Logo compliant applications designed for Windows 2000 need to be designed with some built-in awareness of OnNow. For those who haven't come across this term, it's basically part of Microsoft's Advanced Configuration and Power Interface (ACPI) initiative, which enabled the end-user to treat a PC (often a laptop) as an appliance, putting it into a standby state and then instantly restarting and resuming where he/she left off. In other words, your application needs to be able to go into a state of suspended animation on request, without any loss of data. Part of the design philosophy is that certain key events should be able to 'wake up' an application from its slumbers, a typical example being (for example) the receipt of fax messages during the night. You can find details of OnNow and power management at www.microsoft.com/hwdev/onnow.htm.

An application which needs to continue running, even when the PC is apparently idle, must use the SetThreadExecutionState API call. Thus, Microsoft suggest that video playback or presentation applications can use the ES_DISPLAY_REQUIRED flag in conjunction with the aforementioned API call to prevent Windows 2000 from shutting down the display.

Windows 2000 sends a sleep request to each application by broadcasting a WM_POWERBROADCAST message with wParam set to PBT_APMQUERYSUSPEND. This is basically the operating system saying, 'are you able to go to sleep?' If your application has unsaved data, for example, then it can query bit zero of the lParam field to determine if it's still able to interact with the user (if a laptop's lid has just been closed, then it can't!). If not, then it has to deny the sleep request, otherwise it can display a dialog prompting the user to save the affected document(s).

Conclusions

There are a lot of additional considerations which there's no space to go into here. If you are serious about getting that coveted logo, I'd recommend you download the documents and specifications that I've referred to in this article. Some of the logo requirements have hidden implications and require careful reading. For example, in order to be logo compliant you need to support multiple monitors within your application. Most of the time, this doesn't require any special action on your part, but if you're in the habit of hiding windows by assigning them negative coordinates, for example (a trick that's been around for a number of years), then you may well fall foul of the multiple monitor support because, in a multiple monitor setup, negative screen coordinates may simply make the 'hidden' window reappear on a monitor to the left of the current screen!

The real challenge, of course, is in getting a logo compliant application to work properly under 'lesser' operating systems such as Windows 98 and 95. That's a topic that we may look at in more detail in the future.

Dave Jewell is a freelance consultant, programmer and technical journalist specialising in system-level Windows and DOS work. He is Technical Editor of *Developers Review* which is also published by iTec. You can contact Dave at TechEditor@itecuk.com