Next Stop: Chicago

Writing 32-Bit Applications for All Windows Platforms

by Tammy Steele

Tammy Steele is a technical evangelist in Microsoft's Developer Relations group. When she is not working with software developers on Win32 issues, she wishes it would snow here so she could go skiing.

Later this year, more than four years after the introduction of the Windows 3.0 operating system, Microsoft plans to squire Windows 95 around the personal computer dance floor. No, we're not talking here about a sprawling lakeside urban center somewhere in the midwestern United States, a town with more ward heelers than librarians. We're talking about the next major upgrade of the Windows operating system.

Microsoft is betting that the introduction of Windows 95 will make PCs so much easier and compelling to manipulate that lots more people will want to use them. The result, in turn, will be a greater demand for applications for Windows—and a new, larger market for developers of Windows-based applications. "Windows 95 will bring a fresh new look and new opportunities for applications to become easier to use and more integrated with the Windows platform," said David Cole, Windows 95 group manager.

Start with Win32 and OLE 2.0

"At a high level, there are five key things that developers should incorporate in their Windows-based applications to exploit Windows 95," Cole explained. "The first two are Win32 and OLE 2.0. They are the fundamental building blocks for writing great Windows applications for all platforms. Third, developers should follow the upcoming *User Interface Design Guide* currently being developed for future Windows operating systems, which includes using the new common controls and dialogs.

"The fourth is Plug-and-Play event awareness, which enables customers simply to plug in a device and turn the PC on—the system does the rest. Developers should write applications to be Plug-and-Play event-aware so that they fit in nicely with the Windows 95 system and are aware of changes such as devices coming and going. Finally, your applications should integrate well with the Windows 95 shell by providing registry information, long filename and Universal Naming Convention (UNC) path name support, and supplying file viewers for your data types."

Fortunately, you can exploit all of these five key elements *and* still create applications that run on the Windows NT and Windows 3.1 operating systems with the Win32s application programming interface (API). Not only that, but you will also have an application that runs well on Cairo, the next major release of Windows NT.

Win32: One API, multiple platforms

Using the Win32 API, you can develop an application that runs successfully on Windows 95, Windows NT, and Windows 3.*x*, *and* still take advantage of the unique features of the underlying platform. This is possible because the Win32 API provides an upwardly compatible set of Win32 functions, messages, and structures that behave consistently, whether on Windows 95, Windows NT, Windows 3.*x* with Win32s, or Cairo when it becomes available.

The particular platform's underlying capabilities determine the degree to which the Win32 API is implemented. For example, all Windows platforms, including Windows 3.1 with Win32s, provide an environment for powerful functionality that includes 32-bit sparse memory management (**VirtualAlloc**) and memory-mapped files. Windows 95 and Windows NT allow application writers to take advantage of additional functionality, such as threads and long filenames not available on Windows 3.*x* with Win32s. Windows NT provides additional functionality such as security functions and Unicode API that Windows 95 and Win32s do not.

Because unsupported functions are stubbed on Win32s and Windows 95, you can elect to take advantage of the different levels of functionality in one executable by detecting the underlying platform using the **GetVersion** function. So, for example, you could write an application that uses long filenames and threads when running on Windows 95, Windows NT, and Cairo, but that provides 8.3 names and disabled threading when it is running on Windows 3.*x* with Win32s.

Unlike Windows-based 16-bit applications, Win32 applications running on Windows 95 and Windows NT preemptively multitask, and can run multiple threads of execution in their separate, protected address spaces. Win32 applications allow users to truly perform multiple tasks concurrently. For example, you can run compiles, perform complex spreadsheet recalc, print documents, and play solitaire at the same time.

Portable Win32 apps

From a development perspective, applications gain access to 32-bit flat memory and thus avoid segmentation issues. The Win32 API takes advantage of today's 32-bit processors and allows applications to use new API innovations such as paths, Beziers, and threads. The Win32 API is also portable to non-Intel platforms such as MIPS R4000/R4400, Digital Alpha AXP, and the PowerPC. This means that you simply have to recompile and test an application to enable it to run on multiple hardware configurations!

To top it off, it is relatively easy to port existing Windows 16-bit applications to Win32. In fact, there is a tool that comes with the Win32 Software Development Kit (SDK) called PortTool that scans source code and points out code that needs to change to move from Win16 to Win32. A modified version of PortTool, available in the Microsoft Development Library, not only aids in the porting of applications, but also illustrates how an application can take advantage of the underlying platform and still run across all platforms.

All of the Win32 functions that *both* Win32s and Windows NT support will also be supported by Windows 95. So, if your application runs on both Windows NT and Win32s, you should need to test it only on Windows 95. You can also use the "Writing Great Win32 Applications" table provided in the May release of the Microsoft Development Library. It is an overview of the actual functionality provided by each platform and can aid you in designing and adding features to your applications.

Object linking and embedding

Object Linking and Embedding (OLE) 2.0 gives end users a consistent way to create compound documents, enables cross-application programmability, and provides drag and drop between applications and the desktop. It also allows users to view rich document information, such as icons, actions, and other application-specific properties from the shell.

Windows 95 will be Microsoft's first operating system product to implement the document-centric model for working with personal computers. The development goal for the Windows 95 interface is to lay the basic foundation for the end-user functionality defined by the OLE 2.0 specification and to ensure that current users of Windows are immediately productive when they move to Windows 95

To integrate well with the Windows 95 environment, applications should incorporate OLE 2.0 technologies such as compound files, drag and drop, visual editing, and automation. It is important that your applications use compound files and populate a stream in those files called "Summary Information". Windows 95 will use the information in this stream to give additional information about the file from within the shell. Additionally, if you register the OLE 2.0 OLE class identifier for your application and set the file type properly, Windows 95 can keep track of which application created which document without having to have a file extension associated with it. (See "Just what is an 'OLE application'?" for more specific information on how OLE 2.0 is important to your application.)

User Interface Design Guide: Consistency and integration

One big advantage of the Windows operating system and Windows-based applications is consistency. After learning how to use Windows and at least one Windows-based application, it is relatively easy to get around in a new application. Consistency also builds a cohesive environment and allows people to really focus on their data instead of the application and the shell. There are a couple of tools that will help you provide this consistency in your applications.

One is the *User Interface Design Guide*, which describes how to design software to run on the Windows operating system. Its purpose is to promote visual and functional consistency within and across Windows-based software.

The enhancements in the Windows interface pave the way for the evolution from basic graphical user interface design to a more object-oriented user interface design—a more data-centric, rather than application-centric, interface. The design is a continuation of direction set forth by OLE. As a result, developers and designers may need to rethink their software's interface in terms of what are the basic components and the respective operations and properties that apply to those objects. While applications will still be important, they no longer are the user's primary focus. Users should be able to interact with their data without having to think about starting an application, allowing them to better concentrate on their tasks.

Many of the new visuals will automatically be available to applications making standard Windows API calls. For example, a new title bar with left-justified title text will appear in applications without any modification on the application's part. The same is true of other standard Windows controls. However, if you create an application that uses custom controls, you should take a look at the design guide and consider using the new common controls and dialogs.

Windows 95 will introduce several new Win32 APIs that provide support for common controls and dialogs to make the implementation of new visual user interface features easier. The new common controls include toolbar, status bar, column heading, tabs, slider, progress indicator, rich text control, list view, and tree view. There will also be some modifications made to the existing common dialogs—Open, Save As, and Page Setup—to include additional functionality. Applications already using the common dialogs should continue to do so.

These new common controls and dialogs will be made available for the current versions of Win32s and Windows NT through redistributable dynamic-link libraries (DLLs) at the time Windows 95 ships and, of course, be included on Windows NT Cairo. This way, applications may take advantage of the new controls and dialogs and still run on all Windows platforms.

The revised *User Interface Design Guide,* which you will be able to find in a future release of the Development Library, covers the new controls and dialogs as well as other style guidelines for applications. Additionally, "Seventeen Techniques for Preparing Your 16-bit Applications for Chicago" in the February 1994 *Microsoft Systems Journal* contains an overview of the new controls.

Plug-and-play event-aware: Making PCs easier

Applications should support the new Plug-and-Play messages. Microsoft is working in close cooperation with PC system and component manufacturers to develop a new Plug-and-Play hardware design standard that will enable automatic and dynamic configuration of devices on PC systems. Plug and Play will become a standard feature of Microsoft operating system products, with Windows 95 the first to incorporate full Plug-and-Play functionality.

While Plug and Play might seem like a hardware/system-only initiative, it provides some key functionality that enables applications to respond intelligently to changes in the system. In this way, applications will also integrate well with the Windows 95 system in general. Applications will receive messages when dynamic changes to hardware configuration occur, such as inserting a fax modem or a network adapter, and can automatically take advantage of these new hardware capabilities without requiring any intervention by the user interface. This design makes PCs more intuitive for new and existing users. For example, an application can warn a user about open files on the network if the computer is removed from the network.

Additionally, your application might care about the user changing the resolution of the display on the fly (which can happen on Windows 95). Perhaps you have a different set of icons or bitmaps your application uses depending on the display resolution. When your application gets the WM_DISPLAYCHANGED message, it can update as necessary.

Another important feature, called slow-link awareness, allows your application to be intelligent about remote connections. If the application is running on a machine that is remotely connected to a network via remote access services, the application can be smart about downloading data from the network and autosaving documents.

Applications should also provide a Cancel button for long operations over the network. In the same way, the application can monitor the Advanced Power Management (APM) messages to detect when the system is running on battery. If the system is running on battery, the application can make smart choices about spinning the disk up for background tasks, and autosave frequency.

If you add this functionality, what will happen on Windows NT and Win32s? For the most part, nothing. Windows NT *does* provide support for detecting slow-link, as does Win32s running on Windows for Workgroups if you use the universal thunk to do the same detection. Other than that, current versions of Windows NT and Win32s are not Plug-and-Play aware, so they simply do not broadcast the messages. Windows NT Cairo will be fully Plug-and-Play aware and support all of the Plug-and-Play messages and events.

Shell support

The Windows 95 shell will simplify working with system resources by providing a single tool to work with all resources (documents, programs, printers, utilities, and so on). Windows 95 will unify all the disparate managers—Program Manager, File Manager, Print Manager, Control Panel, Windows Setup—and make it possible to organize all resources in a flexible hierarchy of folders. Drag-and-drop operations will be supported for system resources, and a desktop area will serve as a convenient location to place frequently accessed resources.

Additional ease-of-use enhancements include file viewers to enable users to view file contents whether or not they have installed the application that created the file. Windows 95 will provide some standard file viewers for several data types. However, you will also be able to provide a file viewer for your own data type or a file viewer with more features, such as copy and print, for your data type. Future editions of the Development Library will provide more information about file viewers.

Long filename and UNC path name (\\*servername**share*) support in Windows 95 will make browsing and locating information much easier. Applications should use the UNC path names instead of drive letters so that files opened on a network share can be easily accessed the next time a session is started without requiring the user to reconnect to the same drive letter. Windows NT and Windows for Workgroups 3.11 already support UNC path names. Windows NT and Windows 95 support long filenames, but your applications will see 8.3 names on Win32s. Long filenames and UNC path names are supported by the Win32 API. If you use the common dialogs, you will automatically get long filename and UNC path name support. So all you have to do is make sure your buffer lengths are long enough to store the 255-character filenames plus an additional 260 characters for the UNC path names.

You will also want to register several key items, such as large and small icons, default verbs for context menus, and additional property pages, for your application in the system registry. Windows 95 will use these entries for displaying information about your application. (See "Getting Ready for Chicago" in the May release of the Development Library for additional information.)

Getting on the road to Windows 95

Now that you know some of the important features of a Windows 95 exploitive application, it will be easier to start designing and developing 32-bit Windows 95 applications that also run on Windows NT and on Windows 3.1 with Win32s. A few reminders:

1. See the accompanying reading list.

2. Check out PortTool.

3. Use one of the many Win32 development kits, for example, the Win32 SDK, Visual C++ for Windows NT, Watcom C++ 9.5, Symantec C 6.1, Borland C++ 4.0, or Phar Lap TNT.

4. Design your applications to be Windows 95 exploitive and start developing Win32 OLE 2.0-enabled applications.

5. Test those applications on Windows NT and Win32s today and then on Windows 95, and you will be ready to go with your application.

6. Apply to be added to the Windows 95 beta list by sending e-mail to winbeta@microsoft.com.

PortTool!

Putting on Your Cross-platform Shoes

You can add many features to your applications that will take advantage of the underlying platform. These features include threads, enhanced metafiles, multimedia, communications, networking, advanced graphics, security, and many others. You can use the "Writing Great Win32 Applications" table in the Development Library to help you determine which platform offers what functionality. You can then make choices about what to add to your applications and how to handle running on multiple platforms.

For example, let's look at threads by taking a look at the modified PortTool sample. PortTool is thread-enabled on Windows 95 and on Windows NT, but it has threading disabled when running on Win32s. In the same way, when PortTool is running on Windows 95, it authors a small icon that is displayed by the Windows 95 shell. It also pops up a dialog, when the screen resolution is dynamically changed, to illustrate that an application can respond to a Plug-and-Play event.

A real application, of course, would do something more interesting than popping up a dialog. Furthermore, when PortTool runs on a Windows NT system installed on a dual-processor machine, it automatically gets the advantage of those dual processors, and each thread is leveled across the processors. Additionally, PortTool running on Windows NT and Windows 95 supports long filenames, whereas Win32s uses the 8.3 names. All of this happens with one executable!

In the example below, you can see how easy it is to detect which Windows platform is running your application and take advantage of differences between the platforms. This code snippet from PORTTOOL.C disables the background porting option for PortTool when it is run on Windows 3.1 with Win32s.

```
// Get OS version info, and store in a
// global variable.
 dwVersion = GetVersion();
.
.
// Is there anything specific we need to
// do on different platforms?

if (dwVersion() < 0x80000000)
{
// Windows NT.
}
else if (LOBYTE(LOWORD(dwVersion()))<4)
{
// Win32s.
// Threads aren't available, so disable
// background porting.
EnableMenuItem (GetMenu(hWnd),
    IDM_PORTBKGND, MF_GRAYED);
}
else
{
// Windows 95.
}
```

Full source code for this modified version of PortTool will be available in the May release of the Development Library.

Windows 95 Code Keys

You can get started right away either porting your 16-bit Windows-based applications to Win32 OLE-enabled applications or creating new Win32 OLE-enabled applications with the aid of several tools. With the *User Interface Design Guide*, you can prepare your application to exploit the new Windows

95 user interface by using the existing common dialogs and understanding what the new controls will look like. You can use the "Writing Great Win32 Applications" table to help you understand what functionality the three platforms offer. While you will need to test your application on all targeted platforms, you can be reasonably certain that an application that runs both on Windows NT and on Win32s will also run on Windows 95.

Because there are a few differences among the various platforms, it is extremely important to test on all three. Among the key differences:

- Windows 95 and Windows NT are preemptive multitasking environments that offer separate protected address spaces for all Win32-based applications, whereas Windows 3.1 with Win32s is a non-preemptive multitasking environment in which all applications share the same address space.
- Windows 3.1 with Win32s has a synchronous input queue. Windows 95 and Windows NT have asynchronous input queues.
- Win32s runs on top of Windows 3.1 and therefore has the Windows 3.1 limit of 64K for GDI (graphics device interface) regions. On the other hand, Windows 95 and Windows NT allocate regions out of a 32-bit heap; therefore, regions can be as large as available memory.
- Windows 3.1 with Win32s and Windows 95 have 16-bit world coordinate systems that restrict x and y coordinates for text and graphics to a range of 32K. Windows NT uses a 32-bit world coordinate system allowing a range of 2 GB. If full 32-bit values are passed to text and graphics functions, Win32s and Windows 95 will truncate the upper 16 bits of the coordinates before performing the requested operation.

The "Microsoft Win32 API Overview" included in the May Development Library provides additional information about these and other differences among the three platforms. (Also see "Getting Ready for Windows 95" by Nancy Cluts in the May Development Library.)

Additional Reading

Here is a short reading list that contains important information on writing Win32 applications. Although some of the older articles contain outdated information, they still provide good background.

In the May release of the *Microsoft Development Library*:

- The "Writing Great Win32 Applications" table to determine features to add to your application
- "Getting Ready for Chicago" by Nancy Cluts
- "How to Apply OLE 2.0 Technologies in Applications" by Kraig Brockschmidt
- "Microsoft Win32 API Overview" to help you understand differences between the Windows platforms

In the *Microsoft Developer Network News*:

- "The Win32 story" September 1993

In the *Microsoft Systems Journal*:

- "Seventeen Techniques for Preparing Your 16-bit Applications for Chicago" by Dave Edson, February 1994
- "Windows the Next Generation: An Advance Look at the Architecture of Chicago" by Adrian

King, January 1994

- "Mix 16-bit and 32-bit Code in Your Applications with the Win32s Universal Thunk" by Walter Oney, November 1993
- "At Last—Write Bona Fide 32-bit Programs that Run on Windows 3.1 Using Win32s" by Andrew Schulman, April 1993
- "The Case for 32 Bits" by Charles Petzold, July-August 1992

Top Ten Things to do to Create a Great Windows 95 App

10. Test application on all Windows platforms
9. Register large and small icons, default verbs, and so on in the system registry
8. Monitor Plug-and-Play events
7. Support long filenames and UNC path names
6. Use common controls and dialogs
5. Follow the *User Interface Design Guide*
4. Register **OLEClassID**
3. Populate the Summary Information stream in OLE 2.0 compound files
2. Implement OLE 2.0 drag and drop
1. Use the Win32 API