Microsoft Windows Programmer FAQ

Frequently Asked Questions

Note: *Revision dates for each section are shown next to the section names on each index page! To find updated sections for a particular date, click the* Search *button in* WinHelp *and enter* "Updated:" *to see the various update dates available.*

Microsoft Windows

Windows 1.0	
Windows 2.0	
Windows/386	
Windows 3.0	
Windows 3.1	92-09-21
Windows 3.11	94-03-01
Windows for Workgroups 3.1	93-04-22
Windows for Workgroups 3.11	94-03-01
Windows NT 3.1	94-03-01
Win32s for Windows 3.1	94-03-15
Windows 4.0 ("Chicago" and Win32c)	94-03-15
Windows NT 3.5 ("Daytona")	94-03-15
Windows NT 4.0 ("Cairo")	94-03-15
Windows for Pen Computing 3.1	
Multimedia Windows	
Modular Windows	
Win-OS/2	
Chicago Q&A	94-03-18

Chicago Q&A

Internet and Usenet

Usenet	94-03-15
Usenet Windows newsgroups	94-03-15
Alternatives to Usenet	94-03-02
Windows-related mailing lists	94-03-02
Freeware and shareware by ftp	
Popular Internet ftp sites	93-03-01
Using archie	92-09-21
Ftp by email	
FAQs (Frequently Asked Questions)	93-02-04
More about Internet and Usenet	94-03-15
FTP archives on CD-ROM	92-09-21

Software Development Kits

Microsoft Developer Network Level 2	94-03-10
Windows 3.1 SDK	
Windows 3.1 DDK	
Windows 3.0 SDK	
Windows NT 3.1 DDK	
Windows for Workgroups 3.1 SDK	93-04-25
Win32 SDK	94-03-15
Win32 SDK for Macintosh	94-03-15
LAN Manager Toolkit	93-05-09
MAPI SDK	93-04-25
LSAPI SDK	93-04-25
ODBC SDK	93-04-25
Windows NT SNMP Toolkit	93-04-25

Planning for future versions of Windows

Windows SDK programming techniques

User interface and windows	93-06-20
Dialogs	93-07-30
Controls	94-02-22
Memory	
GDI	93-04-25
Text and fonts	94-02-25
Kernel and low-level programming	94-02-25
OLE and DDE	94-03-19
Miscellaneous	93-05-08

User interface and windows

Dialogs

Controls

Memory

Using new() in C++
Global memory owned by DLL
Determining size of physical memory

GDI

Text and fonts

Kernel and low-level programming

Activating the previous instance	93-01-20
VxD development	94-02-25
VxD technical notes and samples	94-02-25
VxD developer documentation	94-02-25
Getting the instance handle	92-09-28
I/O ports and Windows	94-02-25
Restarting Windows	92-10-05
Rebooting the system	92-10-05

OLE and DDE

Applying OLE technology	94-03-10
OLE resources on ftp.microsoft.com	94-03-19
Using NetDDE	92-12-20

Miscellaneous

Development tool specific issues

Borland ObjectWindows Library

Microsoft Foundation Classes

Turbo Pascal for Windows

Visual Basic

Visual C++

Interfacing to the outside world

Communicating with DOS applications

Multimedia

Miscellaneous

Program Manager DDE interface	93-04-30
Program Manager group file format	93-05-08
TWAIN interface specifications	93-04-30

Putting it all together

Compiling and linking

Debugging

Resources and resource tools

Screen savers

Documentation and help

A programmer's bibliography

Windows 3.1 SDK references	93-04-30
Windows 3.0 SDK references	
Win32 (Windows NT) API references	93-04-30
Windows user interface guidelines	
Microsoft Technical Notes	93-08-13
Programming guides: general	93-07-30
Programming guides: class libraries	93-01-22
Programming guides: device drivers and internals	93-07-30
Programming guides: Visual Basic	93-04-30
Programming guides: macro languages	93-01-22
Magazines	92-09-21
Microsoft Developers' Network	93-04-30
Magazine source code availability	93-04-30

Credits

The author may be contacted by the following means:

Internet:	tomh@ metrics.com
UUCP:	uunet!metrics.com!tomh
BITNET:	tomh@ metrics.com
CompuServe:	>INTERNET: tomh@metrics.com

Mail: Tom Haapanen
Software Metrics Inc.
22 King St. S., suite 303
Waterloo, Ont.
N2J 1N8, Canada

The Word for Windows to Windows Help conversion utility, *Dr. Help,* used for creating and maintaining this document, was created by Roger Hadgraft, senior lecturer in Civil Engineering at Monash University, Clayton, Victoria, Australia. It can be used for converting most Word files into WinHelp files. Roger may be contacted as:

Internet:	roger.hadgraft@eng.monash.edu.au
UUCP:	uunet!eng.monash.edu.au!roger.hadgraft
CompuServe:	>INTERNET: roger.hadgraft@eng.monash.edu.au

I would also like to express my gratitude to the countless people who have contributed information to the Windows FAQs, through Usenet news, email and personal conversations. You know who you are: I'm grateful for your help, as this FAQ would not be what it is without your help.

Latest versions of this FAQ are available by *ftp* on *ftp.metrics.com (198.133.164.1)* in the directory *~/faq.*

Microsoft Windows

Windows 1.0

Microsoft first began development of the Interface Manager (subsequently renamed Microsoft Windows) in September 1981. Although the first prototypes used Multiplan and Word-like menus at the bottom of the screen, the interface was changed in 1982 to use pull-down menus and dialogs, as used on the Xerox Star.

Microsoft finally announced Windows in November 1983, with pressure from just-released VisiOn and impending TopView. This was after the release of the Apple Lisa (but prior to the Macintosh), and before Digital Research announced GEM, another competing graphical environment. Windows promised an easy-to-use graphical interface, device-independent graphics and multitasking support. The development was delayed several times, however, and the first version hit the store shelves (after 55 programmer-years of development!) in November 1985. The selection of applications was sparse, however, and Windows sales were modest,

The following were the major features of Windows 1.0:

- Graphical user interface with drop-down menus, tiled windows and mouse support
- Device-independent screen and printer graphics
- Co-operative multitasking of Windows applications

Windows 2.0

Windows 2.0, introduced in the fall of 1987, provided significant useability improvements to Windows. With the addition of icons and overlapping windows, Windows became a viable environment for development of major applications (such as Excel, Word for Windows, Corel Draw!, Ami, PageMaker and Micrografx Designer), and the sales were spurred by the runtime (Single Application Environment) versions supplied by the independent software vendors. When Windows/386 (see next section) was released, Microsoft renamed Windows to Windows/286 for consistency.

The following are the major changes from earlier versions of Windows:

- Overlapping windows
- PIF files for DOS applications

Windows/386

In late 1987 Microsoft released Windows/386. While it was functionally equivalent to its sibling, Windows/286, in running Windows applications, it provided the capability to run multiple DOS applications simultaneously in the extended memory.

The following are the major changes from earlier versions of Windows:

- Multiple DOS virtual machines with pre-emptive multitasking

Windows 3.0

Microsoft Windows 3.0, released in May, 1990, was a complete overhaul of the Windows environment. With the capability to address memory beyond 640K and a much more powerful user interface, independent software vendors started developing Windows applications with vigor. The powerful new applications helped Microsoft sell more than 10 million copies of Windows, making it the best-selling graphical user interface in the history of computing.

The following are the major changes from earlier versions of Windows:

- Standard (286) mode, with large memory support
- 386 Enhanced mode, with large memory and multiple pre-emptive DOS session support
- No runtime versions available
- Program Manager and File Manager added
- Network support
- Support for more than 16 colors
- API support for combo boxes, hierarchical menus and private .ini files

Windows 3.1 **92-09-21**

Microsoft Windows 3.1, released in April, 1992 provides significant improvements to Windows 3.0. In its first two months on the market, it sold over 3 million copies, including upgrades from Windows 3.0. It is currently continuing to sell at a rate of over 1 million copies per month.

The following are the major changes from Windows 3.0:

- No Real (8086) mode support
- TrueType scalable font support
- Multimedia capability
- Object Linking and Embedding (OLE)
- Application reboot capability
- "Mouse Trails" for easier mouse use with LCD display devices
- Better inter-application protection and better error diagnostics
- API multimedia and networking support
- Source-level API compatability with Windows NT

Windows 3.11 **94-03-01**

Windows 3.11, available now, adds no new features but corrects some existing, mostly network-related problems. It is replacing Windows 3.1 at the retail and OEM levels, and the upgrade is available free from *ftp.microsoft.com*.

Windows for Workgroups 3.1 **93-04-22**

The Windows for Workgroups package, released in November, 1992, is the first integrated Windows and networking package offered by Microsoft. It provides peer-to-peer file and printer sharing capabilities (on a level comparable to LANtastic or Netware Lite) highly integrated into the Windows environment. The simple-to-use-and-install networking allows the user to specify which files on the user's machine should be made accessible to others. The files can then be accessed from other machines running either Windows or DOS.

Windows for Workgroups also includes two additional applications: Microsoft Mail, a network mail package, and Schedule+, a workgroup scheduler.

Windows for Workgroups 3.11 **94-03-01**

Windows for Workgroups 3.11, available now, adds 32-bit file access, fax capabilites and higher performance to Windows for Workgroups 3.1.

Windows NT 3.1 **94-03-01**

Microsoft Windows NT is Microsoft's platform of choice for high-end systems. It is intended for use in network servers, workstations and software development machines; it will *not* replace Windows for DOS. While Windows NT's user interface is very similar to that of Windows 3.1, it is based on an entirely new operating system kernel.

The following are the major changes from Windows 3.1:

- Based on a new microkernel design
- Portable architecture for Intel x86/Pentium, MIPS R4000/R4400 and DEC Alpha processors. Support for PowerPC and SPARC architectures is under development.
- 32-bit addressing for access to up to 4 GB of memory
- Fully protected applications with virtualized hardware access
- Installable APIs for Win32, Win16, MS-DOS, POSIX and OS/2
- Installable file systems, including FAT, HPFS and NTFS
- Built-in networking (LAN Manager and TCP/IP) with remote procedure calls (RPCs)
- Symmetric multiprocessor support
- Security designed in from start, to be initially C2 certified, with a B-level kernel design
- API support for asynchronous message queues, advanced interprocess communication, registration databases, Bezier curves and graphics transformations.

The following is the minimum platform for use with the client edition of Windows NT:

- 33 MHz 386 processor
- 12 MB memory
- 100 MB hard disk
- VGA graphics

The Advanced Server Edition requires 16 MB of memory.

Win32 in itself is not a version of Windows, but the name of application programming interface for Windows NT and Chicago.

Win32s for Windows 3.1 **94-03-15**

Win32s is a set of libraries for Windows 3.1, which enable users to run most Windows NT 32-bit applications on Windows 3.1, without the extensive hardware requirements of Windows NT. The Win32s interface has effectively replaced the older Windows-32 programming interface used by 32-bit Windows applications such as previous versions of Mathematica.

Windows 4.0 ("Chicago" and Win32c) 94-03-15

This unannounced product is rumored to be released in late 1994. It will be a 32-bit system providing full pre-emptive multitasking, advanced filesystems, threading, networking and more. It will include MS-DOS 7.0, but will take over from DOS completely after starting. It will not include Windows NT's security, multiprocessor support, server capabilities or multiple API modules. It will include a completely revised user interface, along the lines of "Cairo", but not taken as far as that product.

See the section entitled **Chicago Q&A** for more information about Chicago.

Windows NT 3.5 ("Daytona") **94-03-15**

"Daytona" is Microsoft's codename for an upcoming release of Windows NT, which will provide OLE 2.0, improved performance and reduced memory requirements. Availability is expected in mid-1994

Windows NT 4.0 ("Cairo") **94-03-15**

"Cairo" is Microsoft's project for object-oriented Windows, and a successor to the Daytona release of Windows NT. Firm details are not available, but most rumors place expected availability sometime in 1995. Developers are encouraged to work with OLE 2.0 in order to start moving in the correct direction towards future "Cairo" compatability.

Windows for Pen Computing 3.1

Microsoft developed Windows for Pen Computing for use on pen-based systems. In most aspects, it is basically equivalent to Windows 3.1 with extensions for pen support. These extensions include the use of a pen as a pointing device as well as handwriting recognition and conversion. Pen Windows first shipped in April, 1992.

Multimedia Windows

The term Multimedia Windows describes a package with Windows 3.0 and the Multimedia Extensions. These extensions are included in Windows 3.1, and thus Multimedia Windows is no longer sold as a separate product.

Modular Windows

Modular Windows is the operating system for Tandy Corp.'s Video Information System (VIS) multimedia player. It is essentially similar to Windows' core, but without any desktop accessories, TrueType fonts or a number of other features.

Win-OS/2

Win-OS/2 is the Windows component of IBM's OS/2 2.0. It is based partially on Windows 3.0 and partially on 3.1. While it runs a majority of the commercial Windows applications, it is not covered by this document.

Chicago Q&A

The following questions and answers are from a document distributed by Microsoft in December, 1993.

Microsoft is continually enhancing its Windows operating system product line to deliver easy to use yet powerful products that exploit the latest advances in microcomputer hardware technology. There is a great deal of interest in and speculation about the Chicago project, the technology development effort which will deliver the next major release of Windows for the mainstream desktop and portable PC. The purpose of this document is to answer the most common questions that customers have voiced about Chicago.

What is Chicago? **94-03-18**

What is Chicago and how does it compare to the Microsoft Windows* 3.1, Windows* for Workgroups and Windows NT* operating systems?*

Microsoft has a family of operating system products designed to fully utilize the range of PC hardware available in the market today, while providing a consistent user interface for end users and a programming environment for developers. Windows 3.x and Windows for Workgroups 3.x on MS-DOS* are designed for mainstream portable and desktop PC platforms. Windows NT is designed for the high-end business and technical workstation platforms and Windows NT Advanced Server is designed as a server platform.

Chicago is the code name for a development project that will produce the successor to Windows 3.x and Windows for Workgroups 3.x. The Chicago project encompasses a variety of important new technologies that will make personal computers running Windows easy to use, and that will provide a more powerful multitasking system and a great platform for communications. Decisions about how those technologies will be packaged will be made later in the development cycle and will be based on customer and business needs.

What is Cairo? **94-03-18**

What is Cairo? How does Chicago compare to Cairo?

Cairo is the code name for a development project that will produce the successor to Windows NT. Chicago and Cairo will produce complementary products that will continue to provide a consistent user interface and programming environment across the entire range of PC hardware platforms.

Why does Microsoft have multiple Windows products? 94-03-18

Why does Microsoft have multiple Windows operating system products? Wouldnt it be simpler to just have one product? Does that mean ISVs have to decide between different operating system products when writing applications?

There are two distinct design points for operating systems platforms. One is centered on the mainstream system, and the other is centered on the high-end system. It is not possible to have one operating system implementation that fully exploits the broad range of hardware available today. At the low end (currently represented by products such as the HP Omnibook and entry-level desktop machines), the primary design goal is to keep the operating system small and fast and to keep usage of machine resources to a minimum. At the high end (for example, a dual-processor technical workstation), the product would need to fully support multiprocessing and advanced 3-D graphics as well as be capable of running technical applications that use maximum machine and system resources.

Over time, low-end machines will become more powerful, and over time, some of todays high-end features will migrate to the low end. In addition, some technical innovations will appear on the mainstream Windows system first, largely because of the timing of product releases, and because some features are focused on end users and ease of use. The Win32 API assures developers that, whichever system they target today, their applications will be able to run in the future as the platform evolves.

Thus, while Chicago and Cairo may leapfrog one another with some features, depending on release cycles e.g., Chicago will sport the next major advance in the user interface, with Cairo inheriting it in its release a few months later the general principle over time is that the high-end product will be a superset of the functionality offered in the mainstream product. Any deviations from this principle are temporary, due to variations in the product release schedules. For ISVs and for development purposes, however, Microsoft has just one Windows platform, defined by the Windows-based 32-bit API, Win32. By following a few simple guidelines, ISVs can write a single application (executable) that runs on the Windows operating system product family. If they wish, ISVs can target specific operating system products because the functionality they provide is important to their particular application, but that is not a requirement. This situation is very much like the Intel microprocessor product line. At any point in time, the Intel product line offers multiple products targeted toward different PC products, ranging from the 80386SL for low-end portable products to the Pentium microprocessor for high-end workstations and application servers. What defines those products is the Intel instruction set, which enables applications to run on all Intel chips, even though the underlying implementation at the transistor level may be very different across the Intel product line. There are also some instructions offered on the Pentium chip that are not on the 80386SL, but ISVs would have to go out of their way to make their products run on only Pentium. And over time, Pentium will become more mainstream, just as the 80486 has become the mainstream microprocessor today, and technologies developed at the low end, such as System Management Mode, will be implemented on the high end as well.

When will Chicago and Cairo ship? **94-03-18**

When will Chicago ship? When will Cairo ship?

Chicago is scheduled to ship in the second half of 1994. Cairo is scheduled to be released in the first half of 1995.

What is Daytona? **94-03-18**

What is Daytona? When will it ship?

Daytona is an interim release of Windows NT that is scheduled to ship this spring.

How will Chicago make the projected ship date? 94-03-18

Major new releases of operating system products have in the past been significantly delayed. How will you make your projected shipment date for Chicago?

Chicago will be released when customers tell us it is ready. The way to make shipment dates is to hit your intermediate milestones. To date, Chicago has been making its milestones with the release of the first Preliminary Developers Kit (PDK) in August and the second PDK in December. Feedback from beta releases beginning in March will tell us more precisely when in the second half of 1994 Chicago will ship.

What if Chicago ships before Cairo? **94-03-18**

If Chicago ships before Cairo, how will users of Windows NT obtain the new functionality in Chicago?

Any new functionality offered in Chicago will be made available to customers of Windows NT through the release of the Cairo product.

What are Chicagos key benefits? 94-03-18

What are the key benefits and features of Chicago? What features will Chicago not have?

For customers, Chicago will present a major step forward in functionality on mainstream desktop platforms by providing a system that is easy to use, offers responsive multitasking performance, and provides a great platform for communications. Ease of use will be delivered through the Plug and Play architecture and an improved, intuitive user interface. Chicago will be a complete, integrated protect-mode operating system that does not require or use a separate version of MS-DOS, implements the Win32* API, and provides pre-emptive multitasking and multiple threads of execution for 32-bit applications. The communications capabilities of Windows will be enhanced with integrated, high-performance networking, built-in messaging, and features such as Remote Network Access and File Synchronization designed for mobile and remote computer users. Chicago will also be a hassle-free upgrade for the current installed base of Windows-based users. Chicago will be compatible with most current applications and drivers for MS-DOS and Windows, and will provide an easy transition to the new user interface features. The applications performance of Chicago will meet or exceed the performance of Windows 3.1 on 80386 systems with 4MB of RAM running the same applications. For systems with more memory, performance will be significantly improved over Windows 3.1. The setup program will enable customers to uninstall Chicago, assuring customers a way to remove it if they are in any way unhappy with it, and will provide tools for system administrators to customize the configuration of Chicago. Chicago will not be processor independent, nor will it support symmetric multiprocessing systems, provide C2-level security, or provide full Unicode support. These features cannot be delivered on the mainstream platform in the near future while still meeting the performance and resource targets necessary to create a compelling upgrade for the huge installed base of users of the Windows operating system. If these features are important to a customer, Windows NT is the product to deploy.

What different Chicago packages will be available? **94-03-18**

What different packages will you have for Chicago?

Decisions about packaging the different technologies being developed as part of the Chicago project will be made later in the development cycle and will be based on customer and business needs. One option is to provide a base Chicago package with some add-on packages that deliver functionality required by specific market segments. This is much like the situation today in which the user of Windows 3.1 can upgrade to Windows for Workgroups by acquiring the add-on package that adds the 32-bit file system and 32-bit networking enhancements to Windows.

What will Chicago be called? **94-03-18**

Since the term Chicago is a code name, what will you call the product(s) that you will eventually release?

Decisions about names will be made after we decide on a packaging plan.

What will happen to MS-DOS? **94-03-18**

What will happen to the MS-DOS product line?

Microsoft will continue to enhance MS-DOS as long as customers require it. Future versions will be derived from the protected-mode technology developed in the Chicago project. Current MS-DOSbased applications and drivers will continue to be compatible with new versions of MS-DOS.

How will Chicago perform on 4 MB? **94-03-18**

Your performance goals on 4MB platforms sound very ambitious, considering all the functionality youre adding to Chicago. How will you achieve those goals?

Chicago will implement new working set management technologies that will optimize the use of memory on low-configuration systems. The networking, disk and paging caches will be fully integrated. Protect-mode device drivers will be dynamically loadable, to ensure that only the drivers that are immediately needed are consuming memory. More components of the base operating system will be pageable. Great attention will be paid to effective page tuning, including hand-tuning source code.

Will Chicago run current applications? **94-03-18**

Will Chicago run my current Windows-based applications? How about MS-DOSbased applications?

Chicago will run most of the current applications for Windows and MS-DOS, as well as new applications written to the Win32 API. Some classes of applications will need to be revised to be compatible with Chicago, such as shell-replacement utilities and file-management utilities. Chicagos new shell provides a complete set of services that is tightly integrated with the operating system components. Shell programs will need to do more than simply replace components such as Program Manager or File Manager. And file-management utility vendors will want to revise their applications to take advantage of the Long File Name feature that Chicago offers. Microsoft is working closely with shell-replacement and file- utility vendors to enable them to revise their products to add value to and be compatible with Chicago.

Will I need to get new device drivers? **94-03-18**

Will I have to get new device drivers to use my current devices with Chicago?

Chicago supports current real-mode device drivers as well as new 32-bit protected mode device drivers. As a result, customers will be able to use their current devices either with their current device drivers, or with new device drivers made available with Chicago. Performance and functionality can be improved if the user installs the new Chicago drivers. Microsoft is making it easier for device manufacturers to deliver new drivers for common devices by defining a more layered, modular device driver architecture. For displays, printers and modems, Microsoft will deliver universal drivers. These drivers will implement common device functionality and expose an interface for device manufacturers to create minidrivers that implement the features specific to their devices. This approach was very successful with printers for Windows 3.1, resulting in rapid availability of fast, high-quality drivers for a wide range of printers.

Will my current applications work well on Chicago? 94-03-18

Will my current applications perform as well on Chicago as they do on Windows 3.1 today?

For Chicago to be a compelling upgrade, Windows-based users must experience a level of performance after installing Chicago that meets or exceeds the performance they currently experience running an identical set of tasks on Windows 3.1. Because a large portion of the installed base of users of Windows today have 4MB systems, Chicago must meet its performance goals on 4MB systems. On systems with more than 4MB of RAM, Chicago will offer significantly improved performance. Understand, however, that there are user and application scenarios today that already use more than 4MB. Users who already require more than 4MB will continue to require more than 4MB with Chicago and if they are using more than 4MB, they should see improved performance. But they wont get away with using less memory in the future than they do today. Its an important distinction to maintain.

When will Chicagos new UI be ready? **94-03-18**

You say Chicago will have a different user interface than Windows and Windows NT. When will that user interface be reflected in the beta versions of Chicago?

The new user interface will be delivered with the first beta of Chicago, scheduled for March 1994.

Will the new user interface mean a lot of retraining? 94-03-18

Wont a new user interface mean a lot of retraining for current Windows-based users? Will the advantages of the new user interface be worth the retraining costs?

The user interface being developed for Chicago will offer dramatic gains in ease of learning and ease of use for the broad range of people using PCs today. Instead of mastering different kinds of tools to work with different resources on their computers, users of Chicago will be able to browse for and access all resources in a consistent fashion with a single tool. This will be much easier than learning separate applications such as Program Manager, File Manager, Print Manager, Control Panel, etc. as users of Windows must do today. A system toolbar that is always accessible will make it much easier to start and switch between full-screen tasks. The implementation of OLE 2.0, with its focus on the users document rather than on the tool used to create it, and the direct manipulation of data through drag and drop in the user interface, will make working with documents easier and more intuitive.

Current users of Windows will be immediately productive with Chicago and be able to learn the new features of the user interface as they work. Chicagos smart setup technology will use the current system settings to present an initial configuration that is familiar for the current Windows-based user. And for corporate customers and individuals who may not want to make any user interface changes initially, Chicago will enable them to continue running their current Program Manager and File Manager configurations.

What is Plug and Play? **94-03-18**

What is Plug and Play? What benefits does Plug and Play provide?

Plug and Play is a technology jointly developed by PC product vendors that will dramatically improve the integration of PC hardware and software. It allows a PC to adapt itself dynamically to its environment; devices can be plugged into or unplugged from a machine, without the user having to do anything special the machine just works. Plug and Play is a general framework that advances that state of the PC architecture by defining how the software communicates with any device connected to the PC. Plug and Play technology enables installation and configuration of add-on devices without user intervention. Plug and Play will make it possible for a consumer to turn a standard desktop system into a great multimedia machine by just plugging in a Plug and Play sound card and CD-ROM, turning on the system, and playing a video clip. Plug and Play can enable new system designs that can be dynamically reconfigured. For example, imagine a docking station that enables you to remove the portable system while it is still running so that you can take it to a meeting, and the system automatically reconfigures to work with a lower-resolution display and adjusts for the absence of the network card and large disk drive. Or imagine an IR-enabled subnotebook that automatically recognizes, installs and configures an IR-enabled printer when you walk into the room, so your applications are ready to print to that printer. Plug and Play can also save development and support costs for the product manufacturer. Today, as many as 50 percent of support calls received by operating system and device manufacturers are related to installation and configuration of devices. With Plug and Play, device driver development is simplified because device manufacturers can write one driver that works across multiple bus types using the Universal Driver Model specified by the Plug and Play architecture. Today, device manufacturers have to include bus-specific code in each of their drivers. With Plug and Play, specific bus configuration data is contained in bus drivers. Also, operating system preinstallation and configuration are simplified for OEMs because Plug and Play devices will automatically install and configure during setup.

What hardware changes does Plug and Play require? 94-03-18

What changes to current hardware and software are required to make Plug and Play a reality? How will vendors figure out how to develop new devices with Plug and Play capability?

First, Plug and Play is compatible with existing systems, so nothing breaks because of Plug and Play. Plug and Play devices can be brought out over time in fact, this is already occurring and will work with existing systems. To deliver all of the above benefits requires changes to devices and drivers, the BIOS, and the operating system. Three fundamental capabilities are required for a system to provide Plug and Play functionality:

- A unique identifier for every device on the system
- A procedure for the BIOS and operating system to install and configure that device
- A mechanism for the system and applications to recognize that a configuration change has occurred while the system is running

All the changes to devices and drivers, the BIOS and the operating system are defined by a series of specifications for Plug and Play architecture. The Plug and Play architecture is an open, flexible and cost-effective framework for designing Plug and Play products. The Plug and Play architecture was jointly developed by a working group of leading vendors, who reviewed design proposals with hundreds of companies in the industry at conferences and through online forums. Plug and Play can be implemented by any operating system vendor and any hardware manufacturer. In addition to Microsoft, IBM has announced support for Plug and Play in OS/2. The Plug and Play architecture is flexible, because it provides a framework that works on multiple types of bus architectures (ISA, SCSI, PCMCIA, VL, PCI, etc.), and it is extensible to future bus designs. The Plug and Play architecture is also cost-effective, because it requires little or no incremental cost for vendors to implement in their products.

Wont it take a long time for Plug and Play? 94-03-18

Wont it take a long time for these changes to be reflected in products? Acceptance of the Plug and Play architecture is widespread, as seen by the rapid progress the industry is making in delivering Plug and Play specifications and products. Specifications have already been released for ISA, SCSI and PCMCIA devices, and the Plug and Play BIOS. Additional specifications are in process, including PCI, ECP, VL, EISA, Micro Channel, and Access. The first Plug and Play devices were demonstrated at COMDEX/Fall 1993, representing a wide range of companies and products. Intel has released development kits that enable device and system vendors to deliver improved configuration capabilities for ISA and PCI systems running with Windows 3.1 in a manner that will provide compatibility with future Windows operating systems. Fully Plug and Play-capable systems (including all Plug and Play devices and a Plug and Play BIOS) will be available in the first half of 1994. These systems will be able to offer complete Plug and Play functionality when combined with Chicago.

Is the Chicago API different from the Windows NT API? 94-03-18

Ive heard that Chicago implements a 32-bit API. Is that API different from the 32-bit API implemented on Windows NT?

There is only one 32-bit Windows API, called Win32, with ISVs able to use the API set to provide different levels of functionality for Windows 3.1, Chicago and Windows NT. Chicago implements a large subset of the functionality of the Win32 API offered on Windows NT, and extends the Win32 API in some areas. These extensions will be delivered on Windows NT as soon as possible after the release of Chicago.

Will vendors need separate Chicago and NT versions? 94-03-18

If there are different implementations of the Win32 API available on different products in the Microsoft operating system product line, does that mean ISVs will have to have separate versions of their applications for Windows and Windows NT?

No. By following some simple guidelines, ISVs can develop a single executable file that runs on Windows 3.x, Chicago and Windows NT. At the recent Professional Developers Conference, we provided in-depth technical sessions on the proper way to design applications to do so, supplied tools in the SDK to help make such development easier, and showed several applications that ran across the entire Windows family.

When will Chicago applications be available? 94-03-18

When will applications be available that exploit Chicago? Wont that take a long time?

ISVs who are developing 32-bit applications for Windows 3.1 and Windows NT using the Win32 API and the guidelines we have provided will have applications that are able to run on Chicago immediately. There are already more than 250 Win32 applications available today, and more coming quickly. Other ISVs will wait until Chicago ships to provide their 32-bit applications; usually those applications start coming on-line about 90 days after the operating system ships. Chicago also will support todays 16-bit applications, so users can move to Chicago immediately and upgrade their applications as they become available. Chicago represents a major market opportunity for ISVs. Chicago will ship on almost all OEM systems soon after it is released, and it will be acquired as an upgrade by a substantial portion of the Windows installed base (the installed base will probably number more than 50 million by mid-1995). Customers who purchase new systems and upgrade their operating systems are the most active purchasers of new software applications. As a result, ISVs have a very significant business incentive to release versions of their applications that exploit Chicago.

Is Chicago completely 32-bit? **94-03-18**

Ive heard Chicago described as a 32-bit operating system, yet Ive also heard that portions of Chicago are implemented with 16-bit code. Are both these statements correct?

Chicago will provide a 32-bit platform for applications by implementing the Win32 API on a complete, protect-mode operating system. Chicago will also run well on mainstream Windows platforms (which for a large portion of the Windows installed base is a 4MB 80386 system), and Chicago will be compatible with applications and drivers for MS-DOS and Windows. These requirements must be met if Chicago is to meet customer needs and provide the volume to make ISVs successful.

These requirements have driven all the design decisions for Chicago. The resulting design deploys 32-bit code wherever it improves performance without sacrificing application compatibility. The design retains existing 16-bit code where it is required to maintain compatibility or where size is a critical issue but has minimal impact on performance. All of the I/O subsystems and device drivers in Chicago, such as networking and file systems, are fully 32-bit as are all the memory management and scheduling components (the kernel and virtual memory manager). Many functions provided by the Graphics Device Interface (GDI) have been moved to 32-bit code, including the spooler and printing subsystem, the rasterizer, and the drawing operations performed by the graphics DIBengine. Much of the window management code (user) remains 16-bit to retain application compatibility.

How do the 16-bit components fit in? **94-03-18**

If portions of Chicago still remain 16-bit, what happens when a 32-bit application makes a function call that is implemented by the 16-bit Chicago component? Doesnt this slow down 32-bit applications on Chicago relative to 16-bit applications?

When Win32-based applications call a 32-bit API that is implemented by a 16-bit component of the system, the function call is translated to its 16-bit equivalent for processing by the system. This translation process is referred to as thunking. Although there is some overhead associated with a thunking operation, the Chicago thunk layer is very efficient. That overhead will be more than offset by the improved efficiency of the linear memory addressing scheme used by Win32-based applications. The overall impact of some thunking code is quite modest vs. all the other work the application and operating system have to do. For end users, perceptions of application performance are based on a combination of the efficiency of the application when executing its own code and the efficiency of the operating system code when the application has called an operating system service. On Chicago systems with adequate memory, end users will experience gains in system efficiency when running 16-bit applications, and they will experience gains in both system and application efficiency when running 32-bit applications.

Will existing networking software work with Chicago? 94-03-18

Will I need new networking software to connect Chicago to my network server?

Customers will require Chicago to connect to their network servers when Chicago is installed, and to offer high-performance, reliable networking functionality. To meet this requirement, Chicago will continue to run existing real-mode networking components. However, we expect customers to want to upgrade to the new 32-bit networking components provided by Chicago. Chicago will enhance the open, flexible, high-performance 32-bit networking architecture offered today with Windows for Workgroups 3.11 that enables customers to mix and match networking components. Chicago will support NDIS 2.0, NDIS 3.0 and ODI drivers, and will provide 32-bit NetBEUI, IPX/SPX and TCP/IP protocols. Redirectors for SMB and NCP-based networks will be included. In addition, Chicagos new multiple-provider interface will make it possible for the user to view, browse and connect to multiple networks in a consistent fashion.

What about Netware with Chicago? **94-03-18**

What about NetWare? Are you working with Novell on NetWare support?

Customers will require high-performance, reliable NetWare support the day Chicago is released. To meet that requirement, Microsoft is developing a 32-bit NCP Redirector that is seamlessly integrated with the Chicago user interface, and is encouraging Novell to do the same. Microsoft will offer Novell access to information and assistance to write a Chicago redirector. Novell engineers attended the Win32 Professional Developers Conference and have been provided access to the Preliminary Developers Kit for Chicago. With this approach, customers should be able to choose from multiple sources for reliable, high-performance NetWare connectivity software when Chicago is released.

Will there be Chicago server version? **94-03-18**

Will there be a Chicago server?

No, not in the sense of a server product such as Windows NT Advanced Server. Chicago will continue to improve upon the peer server capabilities offered in Windows for Workgroups by offering additional features for remote installation, control and administration. These features will make Chicago an even better product for an easy-to-use file and print-sharing LAN that is ideally suited as a small-business, small-department or remote office network. Similarly, Windows NT offers peer services as well for the high-end desktop. But for most server applications, and in the sense that most people ask about a server product, Windows NT Advanced Server is the Microsoft server product.

What about Chicagos portability? 94-03-18

I keep hearing rumors that you are working on a portable version of Chicago. Is this true?

No, we are not working on a portable version of Chicago. Windows NT is our portable operating system, and its already available on high-end Intel, MIPS, Alpha and Clipper machines; it will be available on the PowerPC by mid-1994 and on other high-end platforms over time. There is no reason to make Chicago portable. Chicago is optimized for Intel processors, and much of its internal code is Intel assembler, which puts Chicago at the heart of todays low-end and mainstream line. Portability is important for the new generation of high-powered Intel and RISC machines, on which Windows NT runs and for which Windows NT has been optimized. As these new high-end machines become more mainstream, which will happen over time, Windows NT will already offer the power, security, and reliability that users will demand to exploit these new machines.

What about systems management? **94-03-18**

What will Chicago do to make the client operating system more manageable?

A primary goal for the Chicago project is to make Windows less expensive to deploy in a corporation. Chicago will include some specific features and enabling technologies that will make it easier for system administrators to install, configure, monitor, maintain and troubleshoot their Windows-based desktops. Chicago can be set up from a network server and at the desktop can be configured at the desktop to run locally or across the network. In each case, the administrator can establish a specific configuration for the installation, selecting from a flexible array of setup configuration options. Chicago desktops require only a floppy drive to start up, and paging of components to a swapfile on the network can be disabled to minimize network traffic.

Once Chicago is installed, administrators will be able to centrally configure desktop settings such as file and printer sharing, network access, and passwords. They can remotely monitor Chicago desktops with peer services running to determine what resources are shared, what connections have been made, and what files are being used. Chicago enhances the security provided by Windows for Workgroups to include user-level security. To enable users to access their personal groups, applications, and data from any system on the network, Chicago will provide user profiles. Chicago will also provide the infrastructure for the delivery of enhanced desktop management services by third parties. A backup agent will be included with Chicago to enable administrators to back up desktop data to a network server. To integrate the desktop into SNMP-based enterprise management systems, Chicago will also include a Systems Network Management Protocol (SNMP) agent and a Management Information Base (MIB) for a number of system resources. The system registry and Plug and Play architecture provide a rich store of data about the software and hardware configuration on the desktop, and this information can be accessed by system management software using a DCE-compliant Remote Procedure Call (RPC) mechanism.

Will there be mobility features? **94-03-18**

What improvements will Chicago offer for people who use a mobile or remote computer?

Chicago will provide great support for mobile form-factor devices and will make it easy for end users to access the resources of their desktop systems when they are away from their offices. The implementation of Plug and Play in Chicago will support insertion and removal of devices such as PCMCIA cards while the operating system is running. It will also support automatic reconfiguration of dockable computers when they are inserted or removed from the docking station, without rebooting the system. An enhanced version of Advanced Power Management will further extend battery life. The services provided by Windows for Pen Computing will be enhanced and incorporated into Chicago, including basic inking and rendering support. A special focus will be on remote connectivity. Any Chicago-based machine will be able to serve as a Remote Access dial-up server or a remote client for Windows NT Advanced Server, Novell NetWare servers or Chicago peer servers. The same technology will be used for serial cable and infrared connections between PCs. The Remote Access architecture will be integrated with the Chicago networking architecture by using the same network protocols and advanced security features. Remote Access will support wireless devices and allow application developers to make their applications slow-link aware to improve the user experience when working on a remote system via modem rather than on a high-bandwidth network. Furthermore, Chicago will provide a simple form of file synchronization and APIs for applications to access the file synchronization services to merge changes when both the source document and copy have been modified. Remote e-mail and Microsoft at Work fax capability will be included, as in Windows for Workgroups 3.11 today.

How will file synchronization work? **94-03-18**

Will the file synchronization feature in Chicago provide document management capabilities?

Chicagos file synchronization services are optimized for the needs of the mobile computer user who wants to take copies of documents to a remote location and have them be automatically synchronized with the source documents. It is not intended as a replacement for sophisticated document management systems.

Chicagos file synchronization allows customers to identify files that they want to stay up to date, to change those files, and to have the files automatically updated when the source file is available to the system. The update is performed by replacing the source file with the modified copy at the discretion of the user. If an application writes a merge-handler, then specific data within the modified and source copies of a file can be merged, to create a new updated copy.

Will there be separate NT and Chicago SDKs? 94-03-18

You say you have one API with Win32. Does that mean there will also be just one Windows SDK?

Yes, there will be one Win32 SDK that developers can use to develop 32-bit applications for Windows 3.1, Chicago and Windows NT. In fact, we recently announced a new subscription service, the Microsoft Developer Network Level II that provides developers with not only the Win32 toolkit, but every system toolkit we offer, on a single CD, updated quarterly.

What benefits are there to developers? **94-03-18**

What benefits does Chicago offer to developers? What are you doing to make developing Windows-based applications easier?

The Microsoft Visual Basic programming system has dramatically streamlined and simplified the development of Windows-based applications, and it will be enhanced to support the development of 32-bit applications for Chicago. Microsoft also is enhancing its Visual C++ * development system and Microsoft Foundation Class tools.

Will Visual Basic for Applications be included? **94-03-18**

Will Chicago include Visual Basic for Applications?

Visual Basic for Applications will be offered as a separate product.

Will Chicago and NT use common device drivers? 94-03-18

Will Chicago and Windows NT share the same device drivers?

Generally not, since Chicago and Windows NT have different device driver models. However, since both products support a modular, layered device driver architecture, there are areas of substantial synergy. For example, SCSI miniport adapters for Windows NT will be binary- compatible with Chicago, as will printer drivers and NDIS drivers for Windows NT.

Will WOSA services be included? **94-03-18**

Will WOSA services be included with Chicago?

WOSA is a general, open framework for implementing multiple back-end services in Windows while providing a single front-end interface for end users. Services in Chicago such as messaging and remote network access are designed according to the WOSA framework. Whether or not support for additional WOSA services, such as ODBC support, will be shipped with Chicago is a packaging decision that will be made later in the development cycle and will be based on customer and business needs. All the WOSA- related toolkits are available today to developers through the Microsoft Developer Network Level II subscription service.

Internet and Usenet

Usenet **94-03-15**

If you received this FAQ from somewhere other than Usenet or Internet, you may not be familiar with Usenet. Basically, Usenet is a loose collection of over 1,000,000 computers which exchange mail and news. The network is unstructured and highly distributed; most communication is either by TCP/IP over high-speed connections (or over dual-up links using public telephone lines), or UUCP over public telephone lines.

Internet is the worldwide collection of computers linked using the TCP/IP protocol, consisting of somewhere between 5,000,000 and 10,000,000 computers, usually connected by high-speed TCP/IP network connections.

Usenet *news* is a software system where a person can post an article to a selected newsgroup, and have every other news reader be able to read it. There are over 3,000 newsgroups (including the *alt* groups), and daily volume of news now exceeds 50 MB.

While most Usenet systems are Unix-based, it is not a requirement, and there are a number of Usenet software packages available for Windows as well. If you have an Internet or UUCP connection, ask your system administrator whether you have Usenet news available. Some of the most common newsreading software packages are *readnews*, *rn*, *trn*, *nn* and *notes*.

Usenet Windows newsgroups **94-03-15**

There are a total of eight Usenet newsgroups dealing with Microsoft Windows:

- **comp.os.ms-windows.advocacy**
  This group is intended for adversarial discussions, arguments and comparisons to other computers and operating systems. Applicable to all Windows platforms.
- **comp.os.ms-windows.announce**
  This is a low-volume moderated group with only Windows-related announcements (and the text versions of the FAQs) and with no discussion. Moderated by Steve Graham (*sgraham@shiloh.nimh.nih.gov*).
- **comp.os.ms-windows.apps**
  This group contains discussions, questions, and comments about the selection and use of Windows and Windows NT applications.
- **comp.os.ms-windows.setup**
  This group is meant for questions and discussions about Windows and Windows for Workgroups setup process, driver availability and selection, and hardware compatability and selection.
- **comp.os.ms-windows.misc**
  All other discussions about Windows and Windows for Workgroups should be in this group.

- **comp.os.ms-windows.video** *(proposed)*
  Discussions about vide0 adapters, monitors and video drivers for usee with Microsoft Windows and Windows NT.
- **comp.os.ms-windows.networking.windows** *(proposed)*
  Discussions about Windows built-in networking capabilities: Windows for Workgroups, Windows NT, Windows NT Advanced Server and LAN Manager.
- **comp.os.ms-windows.networking.tcp-ip** *(proposed)*
  Discussions about TCP/IP networking with Windows, WinSock, WinSock-based applications, newsreaders, PPP and SLIP.
- **comp.os.ms-windows.networking.misc** *(proposed)*
  Discussions about Windows and other networks, including Netware, Banyan Vines, LANtastic and LAN Server.

- **comp.os.ms-windows.nt.setup**
  Questions and discussions about the Windows NT setup process, driver availability and selection, and hardware compatability and selection.
- **comp.os.ms-windows.nt.misc**
  All other discussions about Windows NT should be in this group.

- **comp.os.ms-windows.programmer.tools**
  Discussions about the selection and use of tools for Windows software development.
- **comp.os.ms-windows.programmer.win32**
  All discussions about the Win32 applications programming interface (used in Windows NT and Win32s) and the Windows NT SDK belong in this group..
- **comp.os.ms-windows.programmer.misc**
  This group is for all other discussions about Windows software development.

- **comp.os.ms-windows.programmer.bitmaps** *(proposed)*
  Discussions about programming with bitmaps, palettes and DIBs.
- **comp.os.ms-windows.programmer.controls** *(proposed)*
  Discussions about programming with controls, dialogs, custom controls and VBXs.
- **comp.os.ms-windows.programmer. drivers** *(proposed)*
  Discussions about programming Windows and Windows NT drivers and VxDs.

- **comp.os.ms-windows.programmer. graphics** *(proposed)*
  Discussions about programming with graphics, GDI, fonts and printing.
- **comp.os.ms-windows.programmer.memory** *(proposed)*
  Discussions about memory management, processes and DLLs.
- **comp.os.ms-windows.programmer.ole** *(proposed)*
  Discussions about programming with OLE, COM and DDE.
- **comp.os.ms-windows.programmer.winhelp** *(proposed)*
  Discussions about development of WinHelp and MultiMedia viewer applications.

- **comp.binaries.ms-windows**
  This group is for postings of free and shareware Windows applications, utilities, display and printer drivers and for the latest FAQs. Moderated by Tin Le *(tin@saigon.com).*

The following groups have been replaced by those shown above:
- **comp.windows.ms**
  This group was for discussions about Microsoft Windows.
- **comp.windows.ms.programmer**
  This group was for discussions about programming for Microsoft Windows.

The following groups may also be of interest:
- **alt.winsock**
  This group is for discussions about the use and programming of the Windows Sockets interface.
- **comp.databases.access *(proposed)***
  This group is for dicussions about Microsoft's Access database..
- **comp.lang.basic.visual**
  This groups is for Visual Basic (both Windows and MS-DOS versions) discussions.
- **comp.os.msdos.programmer**
  This groups contains general MS-DOS programming questions. Some, especially those concerning compiler selection, may be of interest to Windows programmers.
- **bit.listserv.win3-l**
  This group is a two-way gateway of the BITNET *WIN3-L* mailing list, dealing with all aspects of Windows 3.x.
- **bit.listserv.access-l**
  This group is a two-way gateway of the BITNET *ACCESS-L* mailing list, dealing with Microsoft's Access database.

The following groups are **not** for Microsoft Windows!
- **comp.windows.misc**
  This group is for miscellaneous discussions about windowing systems in general.
- **comp.windows.news**
  This group is for discussions about the Sun Microsystems NeWS windowing system.

In general, these newsgroups are only available to computers connected to Usenet or Internet; they are not gatewayed into BITNET, CompuServe, Prodigy or other services. Some FidoNet BBS systems, however, do carry selected Usenet newsgroups. If you cannot obtain access to these groups on your system, contact the author of this FAQ for possible alternatives.

Alternatives to Usenet 94-03-02

If you are unable to find a connection to the *Internet* (that procedure can not be easily defined, as the *Internet* does not have any sort of a formal structure), there are several alternatives available for finding more information about Windows, and for locating Windows software and drivers.

BITNET users (as well as any other with an electronic mail connection to Internet) can subscribe to lists such as **WIN3-L** *(win3-l@uicvm.bitnet),* a mailing list dedicated to Windows discussions. This mailing list is similar in content to the **comp.os.ms-windows.misc** newsgroup; no programmer mailing list exists on *BITNET*. See the following list for a list of mailing lists.

America OnLine also provides access to Usenet newsgroups.

If you live in North America (or in one of selected Western European countries), you can subscribe to *CompuServe*, a commercial service. *CompuServe* has extensive Windows-oriented discussions and a fairly good selection of free software. Although the level of discussion is often less technical, it is much more structured than the *Internet. CompuServe* also has numerous vendor-supported forums, including ones organized by Microsoft for Windows and Windows NT.

Many *FidoNet*-based BBS systems also carry the Usenet Windows newsgroups. Consult a local BBS listing to find your nearest *FidoNet* BBS.

Windows-related mailing lists 94-03-02

The following mailing lists are Windows-related. Please use the requests address for administrative mail (such as getting added to the list):

- **Dr. Help**
  List: *drhelp@eng.monash.edu.au*
  Requests: *listserv@eng.monash.edu.au*
- **LabView**
  List: *info-labview@pica.army.mil*
  Requests: *info-labview-requesr@pica.army.mil*
- **Lotus Improv**
  List: *improv@bmt.gun.com*
  Requests: *improv-request@bmt.gun.com*
- **MathCAD**
  List: *mathcad@eng.monash.edu.au*
  Requests: *listserv@eng.monash.edu.au*
- **OWL**
  List: *owl-list@cs.rpi.edu*
  Requests: *owl-list@cs.rpi.edu*
- **ProtoGen/ProtoView**
  List: *protoplus@netcom.com*
  Requests: *protoplus-request@netcom.com*
- **WIN3-L** (Windows 3.x)
  List: *win3-l@uicvm.bitnet*
  Requests: *listserv@uicvm.bitnet*

Freeware and shareware by ftp

While CompuServe (which has a lot of software) and your local BBS may have large selections, the Internet provides an immense resource for all PC users. The key program to access this software is called ftp (File Transfer Protocol), and it's usable from most Internet system, but is not usable through UUCP links.

If you do have ftp available to you, follow the example below to connect to ftp.cica.indiana.edu (do not type in the // comments):

```
$ ftp ftp.cica.indiana.edu          // make connection
Connected to ...                    // cica responds
Userid (user@cica): ftp             // enter "ftp" as userid
Password: real_userid@site          // enter your own userid
ftp> tenex                          // for binary transfers
ftp> cd /pub/pc/win3                // where the goodies are
ftp> ls -l                          // list the directory
ftp> get ls-ltR                     // get the current index
ftp> quit                           // we're done!
$ _
```

Of course, you can get multiple files at a time read the ftp manual page for more information.

Remember that **shareware is not free**: register the software you use to encourage the development of more low-cost software.

Popular Internet ftp sites **93-03-01**

The following ftp sites provide significant amounts of software of interest to Windows users:

- **ftp.cica.indiana.edu *(129.79.20.84)***
  Directory */pub/pc/win3* contains one of the largest selections of Windows software and device drivers anywhere. Mirrored by *wuarchive. Please do not access ftp.cica.indiana.edu between 8am and 6pm EST to prevent overloading the system.*
- **wsmr-simtel20.army.mil *(26.2.0.74)***
  Directory *pd1:<msdos>* contains a very large selection of MS-DOS (and some Windows) software. Mirrored by *wuarchive.*
- **wuarchive.wustl.edu *(128.242.135.4)***
  Directory */mirrors/win3* contains a copy of the *cica* Windows archives, and directory */mirrors/msdos* contains a copy of the *simtel10* MS-DOS archive.
- **ftp.uu.net *(137.39.1.9)***
  Directory */vendors/microsoft* contains a lot of the Microsoft developer support materials available on CompuServe, including tech notes, sample sources, the ODBC SDK and WinHelp documentation for Windows and Win32 SDKs.
- **garbo.uwasa.fi *(128.214.12.3)***
  Directories */win3* and */win31* contain a majority of the *cica* Windows archives, and a fair amount of non-*cica* material. *Note that garbo.uwasa.fi is located in Finland, and North American users should avoid congesting transatlantic Internet links by ftping from this site.* Mirrored by *wuarchive.*
- **cc.monash.edu.au**
  Directory */pub/win3* contains a copy of the *cica* Windows archives. *Note that monash is located in Australia, and North American users should avoid congesting transpacific Internet links by ftping from this site.*
- **ftp.und.ac.za**
  Directory */pub/pc/win3/vbasic* contains a a variety of things useful to Visual Basic programmers. *Note that und is located in SouthAfrica, and you should try to avoid congesting transpacific Internet links by excessive ftping from this site.*

If your ftp program complains about an unknown site, you can substitute the numeric Internet address (shown after each site name above) for the name in the ftp command.

Using archie 92-09-21

If you know the program you're looking for, but don't know where to find it, you might try using a utility called *archie.* This program allows you to search for a filename in all the available ftp sites.

There are numerous *archie* servers available; to use one of them, *telnet* to the system, and sign on as *archie*. Follow instructions to search for a file. The following lists some of the know *archie* servers currently available for use; pick one in your geographical area:

· *archie.rutgers.edu*	United States (Northeast)
· *archie.sura.net*	United States (Southeast)
· *archie.unl.edu*	United States (West)
· *archie.mcgill.ca*	Canada
· *archie.au*	Australia and New Zealand
· *archie.funet.fi*	Europe
· *archie.doc.ic.ac.uk*	United Kingdom

Ftp by email

There are several sites that will perform general FTP retrievals for you in response to a similar mail query, although it appears that the *info-server@cs.net* server is permanently out of order.

In general, please be considerate, and don't over-use these services. If people start using them to retrieve megabytes and megabytes of GIF or WAV files, they will probably disappear. Also, keep in mind that your system may be linked to the net using a long-distance UUCP connection, and your sysadmin may not be happy about large mail files using up modem time and filling overloaded spool directories.

- **bitftp@pucc.bitnet**
  For information on this one (available only to BITNET sites) send it the message:
  help

- **ftpmail@decwrl.dec.com**
  For information on this server, available to all Internet sites, send it a mail message with a body containing simply:
  help

- **mailserv@garbo.uwasa.fi**
  One final choice is to use the *garbo.uwasa.fi* server, which lets you access the *garbo.uwasa.fi* archive (which contains most of the *cica* files). For instructions, send it a mail message with "*Subject: garbo-request*" and a single line of text "send help" to
  send help
  Please do not use this service if you are located in North America!

FAQs (Frequently Asked Questions) **93-02-04**

Hundreds of Usenet newsgroups have their own FAQs, most of them in text format. You can retrieve almost all of these FAQs' latest versions by ftp from *rtfm.mit.edu* in the directory */pub/usenet/news-answers.*

More about Internet and Usenet 94-03-15

To learn more about Internet and Usenet, I strongly recommend you purchase or borrow a copy of Ed Krol's *The Whole Internet User's Guide and Catalog* (ISBN 1-56592-025-2, $24.95), which covers email, news, ftp, archie and much more. This 400-page handbook is a thorough guide to getting around on the Net, clear enough for neophytes but with new information even for true Internet veterans. A wide range of other books are also available; check your local bookstore for the selection.

To purchase *The Whole Internet User's Guide and Catalog*, check your local bookstore or contact the publisher, O'Reilly and Associates at 1-800-998-9938 (103 Morris St., Sebastopol, CA 95472).

FTP archives on CD-ROM **92-09-21**

Walnut Creek offers copies of the *cica, wuarchive* and *simtel* FTP archives on CD-ROM, at prices ranging from $25 to $50, with annual subscriptions available. Call (800) 786-9907 or (510) 947-5996 for more information.

Software Development Kits

Microsoft Developer Network Level 2 **94-03-10**

The preferred way to get all the SDKs listed in the subsequent sections is to subscribe to the Microsoft Developer Network, Level 2. For $495/year you get not only the extremely useful MSDN CD-ROM discs four times a year, as well as quarterly updates of *all* the SDKs and DDKs available from Microsoft.

Windows 3.1 SDK

The primary Windows development tool you need to do development is the Windows 3.1 Software Development Kit (SDK). It includes the libraries, header files, resource tools, documentation and the Windows debug kernel you need to create native Windows applications.

Before you rush out to buy the SDK, though, note the following points:

- A number of integrated development tools (such as Actor, Visual Basic and Turbo Pascal for Windows) do not require the SDK to operate.

- A number of compilers (such as Microsoft C/C++ 7.0, Borland C++ 3.0 and Zortech C++ 3.0) include the SDK to operate.

The SDK includes the tools you need to create pen-based and multimedia applications, and it also allows you to create applications to run on Windows 3.0.

The list price of the Windows 3.1 SDK is $349.

Windows 3.1 DDK

In order to develop device drivers or VxDs for Windows 3.x, you need to purchase the Windows 3.1 Device Driver Kit (DDK). It includes the necessary header files, libraries, documentation and sample source code to create new device drivers.

The list price of the Windows 3.1 DDK is $495.

Windows 3.0 SDK

The older version of the SDK, 3.0, is still quite useable with Windows 3.1, although it includes older versions of the resource tools, Windows 3.0 debug kernel and is not capable of creating applications which take advantage of the new Windows 3.1 features.

Windows NT 3.1 DDK

The Windows NT Device Driver Kit (DDK) is be available from Microsoft for $495, including full printed documentation, on CD-ROM only.

Windows for Workgroups 3.1 SDK **93-04-25**

The Windows for Workgroups SDK is intended for creating workgroup applications using the new APIs introduced in Windows for Workgroups. This SDK requires that you already have the Windows 3.1 SDK (or equivalent).

The Windows for Workgroups SDK is available free of charge from Compuserve, or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/wfwsdk.*

Win32 SDK **94-03-15**

The latest member of the Windows SDK family is the Win32 Software Development Kit (SDK). The Win32 SDK also includes the tools for creating Win32s applications.

Win32 SDK for Macintosh **94-03-15**

The Win32 SDK for Macintosh allows developers to create native Macintosh-based applications from existing Win32 or MFC applications. It implements most of the Win32 interface using the standard Macintosh facilities, and includes a cross-compiler for the Motorola 68000 (a PowerPC version is also forthcoming).

LAN Manager Toolkit **93-05-09**

The LAN Manager toolkit is intended for creating Windows and Windows NT applications that want to access LAN Manager and/or Windows NT network functionality. This SDK requires that you already have the Windows NT SDK.

The LMAPI SDK is available free of charge from Compuserve (except for connection charges), or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/LMAPI.*

MAPI SDK **93-04-25**

The MAPI SDK is intended for creating mail-aware and mail-enabled applications using the MAPI interface promoted by Microsoft. This SDK requires that you already have the Windows 3.1 SDK (or equivalent).

The MAPI SDK is available free of charge from Compuserve (except for connection charges), or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/MAPI.*

LSAPI SDK **93-04-25**

The LSAPI SDK is intended for creating license-controlled network applications for Windows, using LSAPI, the new industry standard licensing API. This SDK requires that you already have the Windows 3.1 SDK (or equivalent).

The LSAPI SDK is available free of charge from Compuserve (except for connection charges), or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/LSAPI.*

ODBC SDK **93-04-25**

The ODBC (Open DataBase Connectivity) SDK is intended for accessing a variety of different database formats using a standardized SQL-based API. This SDK requires that you already have the Windows 3.1 SDK (or equivalent).

The ODBC SDK is available free of charge from Compuserve (except for connection charges), or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/odbc-sdk.*

Windows NT SNMP Toolkit **93-04-25**

The SNMP SDK is intended for creating Windows NT applications that access mail through SNMP (Simplke Network Mail Protocol). This SDK requires that you already have the Windows NT SDK.

The LSAPI SDK is available free of charge from Compuserve (except for connection charges), or by *ftp* from *ftp.uu.net* in the directory */vendor/microsoft/LSAPI.*

Planning for future versions of Windows

Application Compatibility in Future Versions of Windows 93-05-08

Courtesy of Microsoft Corporation
Created: February 12, 1993

Abstract

This article provides guidelines for writing applications for the MicrosoftÒ Windows version 3.x operating system in a manner that will produce the fewest compatibility problems when the application is run on future versions of Windows. The discussion focuses on compatibility issues involving Windows-based applications, Windows display drivers, and MS-DOSÒ-based applications.

Guidelines for Windows-based Application Developers

Keep these general rules in mind when developing applications for Microsoft Windows :

- The golden rule of application compatibility is to adhere to the Microsoft Windows Software Development Kit (SDK) documentation. That is, don't use an application programming interface (API) that is not documented, and only use the features of an API that are documented.

- Don't depend on the format of internal data structures to remain the same in the future. For example, the format of the internal structures used for windows (HWND), menus (HMENU), device contexts (HDC), regions (HRGN), bitmaps (HBITMAP), and tasks (HTASK) are guaranteed to change in a future version of Windows. Other internal structures may also change.

- Don't assume objects are allocated in USER's or GDI's data segment. In an attempt to remove system resource limitations, objects that are currently allocated in these data segments may be allocated elsewhere in the future. For example, assuming a window handle is an offset in USER's data segment would probably be incorrect in future versions of Windows.

- Don't replace system DLLs such as TOOLHELP.DLL, SHELL.DLL, and COMMDLG.DLL unless you use the version APIs (VER.DLL). These DLLs will change in the future. The system will malfunction if applications replace these DLLs with older 3.0 or 3.1 versions. If your application installs these DLLs, double-check the code for correctness; many applications that attempt this do not do it correctly.

- Test the Windows version number properly. The following code, for example, will not work correctly if it is run on a version of Windows that is numbered 4.0 because the first test of the minor version will fail. Surprisingly, this is a very common mistake.

  ```
  winVer = LOWORD(GetVersion());
  if (HIGHBYTE(winVer) >= 10 && LOWBYTE(winVer) >= 3)
      // run
  else
      //exit
  ```

  Use the following code instead:

  ```
  winVer = LOWORD(GetVersion());
  ```

```
winVer = (((WORD)(LOBYTE(winVer))) << 8)|(WORD)HIBYTE(winVer);

// NOTE: Always use a HEX value here!!!
if (winVer >= 0x030A)
     // run
else
     //exit
```

- Applications written for Windows version 2.x will not be supported under future versions of Windows. Make sure your applications have been tested and built using any of the Windows version 3.x SDKs so that they are marked as applications written for Windows version 3.0 or higher and can run in protected mode.

- Don't copy Program Manager group files onto a user's disk. Use the Program Manager's dynamic data exchange (DDE) interface to add groups and group items for your application.

- Don't assume minimized application windows have icon title windows. If your application walks the window list and assumes that windows with a class name of "0x8004" or "#32772" are icon titles, the application will not function properly in future versions of Windows. If your application needs to perform this operation when running on Windows version 3.1, write your code so that the application will continue to work even if it doesn't find the icon title windows.

- Don't hard-code the pixel dimensions of menus, scroll bars, sizes of captions, and such. Instead, use GetSystemMetrics to get these sizes. The sizes will change depending on the active display driver and may be user-adjustable in the future. Also, your code should watch for the WM_WININICHANGED message and reinitialize the values accordingly.

- Don't hard-code button colors to be the standard three shades of gray. Use the GetSystemColors function to obtain these colors. Again, watch for the WM_WININICHANGED message, and reinitialize these colors accordingly.

- Those writing debuggers must use the services provided by TOOLHELP.DLL, rather than the services provided by the older WINDEBUG.DLL. WINDEBUG.DLL will not work in future versions of Windows.

- Don't assume that GlobalWire allocates MS-DOS addressable memory. Your application must use GlobalDOSAlloc to obtain this type of memory.

- Don't assume that GlobalAlloc with the GMEM_FIXED option allocates MS-DOS addressable memory. Your application must use GlobalDOSAlloc to obtain this type of memory.

- Printer soft font information is currently stored in WIN.INI and is associated with a particular port (LPT1, for example). In the future, this information will be associated with a printer in order to be independent of the port to which the printer is connected.

- Your application must not assume the contents of any WINOLDAP (MS- DOS application manager) data structures allocated in WINOLDAP's data segment. These structures may change in the future.

- Do not overtune your application's STACKSIZE or HEAPSIZE settings in the application's .DEF file. Some developers have tuned these settings (STACKSIZE, in particular) in their applications to supply exactly enough space to run on Windows version 3.0 or 3.1. These applications sometimes have problems because different Windows display drivers have different stack depth characteristics. Future versions of Windows will compound this problem because the stack depth will change for most of the core components (GDI, KERNEL, USER, and so forth). It is recommended that at least an additional 2K be added to the minimum STACKSIZE and HEAPSIZE settings.

Guidelines for Display Driver Developers

Keep these points in mind when developing display drivers for Windows:

- The meaning of the WindHand field in the EXTPAINTSTRUC may be changed for enhanced mode grabbers. WindHand is the HWND of the grabber child window inside the WINOLDAP window. All grabber painting should be restricted to this window. Grabbers weren't supposed to use WindHand for anything beyond calling GetClientRect, GetDC, and such.

- Grabbers shouldn't use the EPStatusFlags bits other than fFocus, fVValid, fSelect, and fGrbProb. Some bits that are private to WINOLDAPP were accidentally included in the DDK header files although not used in any Microsoft-distributed grabber sample source.

Guidelines for MS-DOS Application Developers

If you develop applications for MS-DOS, keep these rules in mind:

- Make sure your application works properly in a Windows version 3.1 MS-DOS box. Especially make sure your setup program will function in a Windows MS-DOS box. For example, writing over Program Manager group files or altering WIN.INI or SYSTEM.INI while Windows is running would be bad things to do. Even though the application is MS-DOS-based, consider writing a Windows-based setup program, especially if your setup process needs to perform operations such as altering WIN.INI or SYSTEM.INI.

- Don't assume the location of the system file table (SFT) or MS-DOS buffers. These may be moved into high memory to provide extra conventional memory. In general, all internal MS-DOS data structures may be moved into high memory in the future.

- Don't assume sizes of internal MS-DOS data structures. For example, don't assume that a drive parameter block (DPB) is 21h bytes long as some applications have. The format of data structures-such as these that are easy to find and traverse-very well may change in future versions of MS-DOS. Use documented INT 21h calls to obtain information such as this. For example, DPBs can be obtained using INT 21h functions, 1Fh and 32h.

Preparing your application for Chicago 94-03-19

This section is based on the information presented by Dave Edson in the February 1994 issue of Microsoft Systems Journal. Consult this issue for further details.

- *Use GetSystemMetrics API for everything it can be used for*
  Chicago allows users to customize the system extensively; you should make certain that you use the correct parameters and attributes.
- *Process WM_WININICHANGED*
  Make sure you handle changed attributes.
- *Avoid drawing non-client areas yourself*
  Your look might not conform to Chicagos.
- *Dont draw on the icons*
  Chicago will use a different API for drawing into the icon.
- *Use only TrueType fonts in dialogs*
  Chicago will allow users to scale the dialogs.
- *Use VER.DLL in your installation process*
  Make sure you dont overwrite newer DLLs during the installation.
- *Use compound files for storage*
  Using OLE 2s STORAGE.DLL will allow you to save extra information (such as icon and description) in the file, which will then be accessible to Chicagos shell.
- *Remember general portability*
  Consult the preceding section for general portability guidelines.
- *Beware of multiple instances*
  Chicago wont restrict multiple instances of applications; make sure you handle such situations in a reasonable fashion.
- *Use common dialogs*
  Get the Chicago look and feel for free!
- *Support OLE 2 drag and drop*
  This will integrate your application into Chicagos desktop.
- *Increase your stack and heap allocation*
  32-bit values need more space than 16-bit ones.
- *Dont draw on the icons*
  Chicago will use a different API for drawing into the icon.
- *Allow for long filenames*
  Allow for filenames of up to 254 characters (this is also the maximum length for a pathname), even if you dont do any specific processing for them.

Things youll add later for your Chicago app 94-03-19

This section is also based on the information presented by Dave Edson in the February 1994 issue of Microsoft Systems Journal. Consult this issue for further details.

- *Support Chicago-style help*
  This is rumored to be based on the Multimedia Viewer.
- *Use threads, memory-mapped files and asynchronous I/O*
  You can start using these today with Win32 for Windows NT.
- *Support pen input*
  Again, you can support Windows for Pen in your current applications; Chicago will build on this technology.
- *Use new controls*
  Chicago will support several new controls. Depending on your perspective, you can currently roll your own, use a 3rd-party control, or simulate their behavior using existing Windows controls. In any case, use an insulating layer so that our app wont need to be changed once you switch to Chicagos native controls.
  - *Slider*
    You can get the feel, if not the look, using a scrollbar.
  - *Progress Meter*
    Use a 3rd-party control, or implement it yourself.
  - *Spin button*
    Use Windows 3.1 SDKs MUSCROLL.
  - *Toolbar*
    Use the MFC toolbar, a 3rd-party control, or roll your own.
  - *Status bar*
    Use the MFC toolbar, a 3rd-party control, or roll your own.
  - *ListView*
    Use a 3rd-party control, or implement it as an owner-draw listbox.
  - *Column Heading*
    Use a 3rd-party spreadsheet control.
  - *Property sheets and folder tabs*
    Study Word 6 or Escel 5, and implement your own; really, these are quite simple, and yet add significant pizzazz to your application.
  - *Rich Text edit control*
    Use a 3rd-party control.
- *Replace Cut/Copy/Paste with Move to Here/Copy to Here*
  Use the right mouse button to pop up these new actions.

Windows SDK programming techniques

User interface and windows

Activating a window without bringing it to the top **93-06-20**

In Windows 3.1 and Windows NT, it is possible to activate a window without making it the topmost one (i.e. without changing the existing window Z order). To do this, make the following function call:

```
SetWindowPos(hWnd, NULL, 0, 0, 0, 0,
        SWP_NOMOVE | SWP_NOSIZE | SWP_NOZORDER);
```

You will then need to process the WM_WINPOSCHANGING message, and clear the SWP_NOZORDER flag *again*, this time in the WINDOWPOS structure.

Animating the cursor **93-04-25**

While Windows NT provides an API for animating cursors, Windows 3.x does not. If you have hte need for animated cursor, you can use WM_TIMER to generate timer messages at fixed intervals, and then load the next cursor resource whenever you receive a WM_TIMER message.

Note that cursor animation may, depending on the user's display hardware, cause quite noticeable flicker; you should probably provide an option for the user to disable the cursor animation.

Changing the icon on the fly **92-11-04**

If you want to change the icon of an application while it is minimized, you need to call:

```
SetClassWord(hWnd, GCW_HICON, hIcon);
RedrawWindow(hWnd, NULL, NULL, RDW_FRAME | RDW_ERASE |
        RDW_INVALIDATE);
```

This call with these flags invalidates a window's non-client area, with the RDW_ERASE forcing a WM_ERASEBKGND.

Changing the application's language **93-01-20**

Windows is designed to make it easy for the developer to maintain multiple national language versions of an application. The general technique involves modifying only the resource file (with menus, dialogs, strings and accelerators) for each national edition, and leaving the source code alone. For example, to change the message strings to the correct national language, you'd have multiple **.rc** files, one for each language, such as this:

rc.h

```
#define IDS_NOTFOUND       8000
#define IDS_REPLACEFILE   8001
...
```

english.rc

```
STRINGTABLE
BEGIN
    IDS_NOTFOUND,          "File not found"
    IDS_REPLACEFILE,      "Replace existing file '%s'?"
    ...
END
```

finnish.rc

```
STRINGTABLE
BEGIN
    IDS_NOTFOUND,          "Tiedostoa ei löydy"
    IDS_REPLACEFILE,      "Korvaa tiedosto '%s' uudella?"
    ...
END
```

error.c

```
DisplayError(WORD wCode)
{
    char szMsg[256];

    LoadString(hInst, wCode, szMsg, 256);
    MessageBox( ..... );
}
```

For each national language version, you would use *rc* to compile the resources, and then build them into the executable.

If you need to be able to switch languages on the fly, you have several options. First, you can use multiple DLLs, each one containing a particular language's resources. As an alternative, if the number of languages you need to support is fairly limited, you can include all the languages you need to support. In this example, you would have something like:

rc.h

```
#define IDL_ENGLISH          0
#define IDL_FINNISH        100
#define IDL_SWEDISH        200
...

#define IDS_NOTFOUND       8000
```

```
#define IDS_REPLACEFILE      8001
...
#define IDS_E_NOTFOUND       (IDS_NOTFOUND      + IDL_ENGLISH)
#define IDS_E_REPLACEFILE    (IDS_REPLACEFILE + IDL_ENGLISH)
...
#define IDS_F_NOTFOUND       (IDS_NOTFOUND      + IDL_FINNISH)
#define IDS_F_REPLACEFILE    (IDS_REPLACEFILE + IDL_FINNISH)
...
```

national.rc

```
#include <english.rc>
#include <finnish.rc>
...
```

english.rc

```
STRINGTABLE
BEGIN
    IDS_E_NOTFOUND,          "File not found"
    IDS_E_REPLACEFILE,      "Replace existing file '%s'?"
    ...
END
```

finnish.rc

```
STRINGTABLE
BEGIN
    IDS_F_NOTFOUND,          "Tiedostoa ei löydy"
    IDS_F_REPLACEFILE,      "Korvaa tiedosto '%s' uudella?"
    ...
END
```

error.c

```
DisplayError(WORD wCode)
{
    char szMsg[256];
    extern WORD wLanguage;

    LoadString(hInst, wCode + wLanguage, szMsg, 256);
    MessageBox( ..... );
}
```

Changing the restored size of a maximized window **93-01-20**

If your application's window is maximized, and you want it to be set to a size other than what it was (before maximizing) when the user hits the "restore" button, you will need to process the WM_WINDOWPOSCHANGING message in your window procedure. This will allow you to set the size and position of the restored window to anything you need.

Creating an initially invisible MDI child window

Before creating the child window,

```
SendMessage( hClientWindow, WM_SETREDRAW, 0, 0L);
```

Then, in your child window WM_CREATE processing,

```
ShowWindow( hChildWindow, SW_HIDE).
SendMessage( hClientWindow, WM_SETREDRAW, 1, 0L);
```

Drag-and-drop: File Manager and Print Manager

You will need to register your application in the registration database. You can do this either using the Registration Editor, or the Reg* API in Windows 3.1 SDK. One of the simplest mechanisms is that used by several Windows 3.1 applets to print a file the parameters are

/p filename

See the registration database for examples.

Drag-and-drop: generalized client **93-01-20**

To do generalized drag-and-drop, you'll need **shell.dll**, shipped with Windows 3.1.

- Either do DragAccept() or create the window as WS_EX_DROPFILES (0x10L)
- Wait for the WM_DROPFILES message (0x0233), which passes a handle to something in wParam
- You can then issue
  WORD DragQueryFile(hDrop, 0xffff, NULL, 0)
  to get the file count, and then
  WORD DragQueryFile(HANDLE hDrop, WORD nFile, LPSTR sDest, WORD max)
  for each of the dropped files
- Once you have finished, call
  DragFinish(hDrop)

For Visual Basic, get the file **dd.zip** from *CompuServe*'s MBASIC forum, which lets you implement drag-and-drop from VB. This file may also be available at *ftp.uu.net* and *ftp.cica.indiana.edu.*

Drag-and-drop: generalized server **93-01-20**

To act as a drag-and-drop server (meaning that the user can drag files from your application to a drag-and-drop client application), you will need to follow the following steps:

- Capture the mouse
- On mouse movement, get the client window with WindowFromPoint(), and test it for WS_EX_ACCEPTFILES style to determine whether to allow dropping
- Construct a file structure in memory allocated with GMEM_DDESHARE as follows:

```
typedef struct tagDRAGHEADER {
   UINT wStructSize;         // Size of the structure (8 bytes)
   UINT x;                   // x-coordinate in window's client hDC
   UINT x;                   // y-coordinate in window's client hDC
   BOOL fInClient;           // TRUE if on window's client area
} DRAGHEADER;
```

  followed by a list of filenames separated by nulls, and terminated by a double null.

- Post a WM_DROPFILES message to the client window with the above structure as a parameter.

Forcing a window to stay fixed size or iconic

In order to make your app stay as an icon, you must process the WM_QUERYOPEN message. If you always return 0 for this message, you indicate that the icon can not be opened into a ordinary window.

To retain a fixed size, you must process the WM_GETMINMAXINFO message. When you get it, modify the info pointed to by lParam:

```
LPPOINT lpSize = (LPPOINT)lParam;
lpSize[3].x = lpSize[4].x = theRightWidth;
lpSize[3].y = lpSize[4].y = theRightHeight;
```

If you don't want the window to be maximized or iconized, create it with the ~WS_MAXIMIZEBOX and/or ~WS_MINIMIZEBOX styles, and disable those items from the system menu, if there is one.

Also, you can alternately disable resizing by creating the windows with ~WS_THICKFRAME, and disabling the Size... item on the system menu.

Getting the handle of the active window **93-01-20**

To get the active window's handle, just call

```
hActiveWindow = GetFocus();
```

Keeeping a window on top **93-01-20**

To keep a window on top of all other windows without requiring it to be the active window (akin to the behaviour exhibited by the Windows 3.1 clock), you can make the following function call:

```
SetWindowPos(hWnd, (HWND)-1, 0, 0, 0, 0,
        SWP_NOMOVE | SWP_NOSIZE | SWP_NOACTIVATE);
```

Note that it is not considered polite behavior by an application to always keep its window topmost, unless you at least give the user the option of disabling this "feature".

Limiting window size **92-12-15**

To limit window size, you'll need to process the WM_GETMINMAXINFO message, and fill in the structure describing the minimum and maximum window sizes.

Right-justifying menu items

To right-justify an entire menu item or just a part of it, place a \a in the string just before the right-justified part.

Incidentally, the Windows 3.0 CUA guidelines no longer call for right-justifying the Help menu on the menu bar.

Right-justifying menu items at runtime

It's undocumented, but what you need is a 0x08 in the menu string. The easiest way to do this is to place a \b in the string before the right-justified part (either the text of the accelerator key).

Incidentally, the Windows 3.0 CUA guidelines no longer call for rightjustifying the Help menu on the menu bar.

Trapping mouse clicks on desktop

To trap mouseclicks on the desktop, you will need to subclass the desktop window. The following code fragment illustrates the technique (sample code courtesy of Blake Coverett, *blakeco@microsoft.com*):

To subclass the desktop:

```
hWndDesktop=GetDesktopWindow();
hSaveTask=GetCurrentTask();
lpfnDesktop=(FARPROC)GetWindowLong(hWndDesktop, GWL_WNDPROC);
lpfnSubClassProc=MakeProcInstance((FARPROC)WndProc, hInst);
SetWindowLong(hWndDesktop, GWL_WNDPROC,
                (LPARAM)(LONG)lpfnSubClassProc);
```

and then to undo it when ready to unload:

```
SetWindowLong(hWnd,GWL_WNDPROC, (LPARAM)(LONG)lpfnDesktop);
PostAppMessage(hSaveTask,WM_QUIT,0,0);
```

Using status bars with MDI **92-09-15**

Add the following code fragments to the indicated places in the WinProc() of an application, or the FrameWinProc() of a MDI application.

```
case WM_CREATE:
    hWndClient = CreateWindow( "MDIClient",...,
                    WS_CHILD|WS_VISIBLE|WS_CLIBSIBLINGS|
                    WS_HSCROLL|WS_VSCROLL,... );
    hWndStatus = CreateWindow( "Static",...,
                    WS_CHILD|WS_VISIBLE|SS_LEFT|SS_NOPREFIX,... );
    ...
case WM_SIZE:
    GetClientRect( hWnd,&rect );
    // Calculate DIVIDING_LINE such that.
    // rect.top < DIVIDING_LINE < rect.bottom
    // One choice:
    // DIVIDING_LINE = rect.bottom - GetSystemMetrics(SM_CYMENU);
    MoveWindow( hWndClient,rect.left,rect.top,
                rect.right,DIVIDING_LINE,TRUE );
    MoveWindow( hWndStatus,rect.left,DIVIDING_LINE,
                rect.right,rect.bottom,TRUE );
    break;   // Do *not* pass to DefFrameProc() of MDI app!!!
    ...
    // To change the status text:
    SendMessage( hWndStatus,WM_SETTEXT,0,(LONG)(LPSTR)szStatusText );
```

Notes:

- For non-MDI applications, all references to hWndClient above will simply be removed.
- Menu selection can be tracked by setting the status text at a response to the WM_MENUSELECT message.
- To draw a line between status bar and the rest of the client area, the DIVIDING_LINE should be adjusted in either MoveWindow() call to leave a gap between, which is called InvalidateRect() for, and actually being painted in response to the WM_PAINT.
- The parent window should have WS_CLIPCHILDREN style bit set.

Dialogs

Adding controls to a non-dialog window

You can do this by simply calling CreateWindow() with one of the predefined child window control class names (see the control class definition table in the SDK reference manual).

Creating 3-D "look" dialogs **93-04-22**

The current Microsoft-endorsed method of creating dialogs with the chiseled 3-D look is to use the **ctl3d.dll** dialog, which gives you a look similar to Excel 4.0's dialogs with very low effort. This library is included on the Microsoft Developer Network CDs, and is also available by *ftp* from *ftp.uu.net,* directory */vendor/microsoft/developer-network.*

Doing a timeout in a dialog

Start a timer in your WM_INITDIALOG processing. If your dialog box receives the WM_TIMER message, kill the timer and post yourself a WM_COMMAND messgae with wParam == IDOK. If the user presses any button, restart the timer.

Minimize button on modal dialog moves when clicked

It's a bug in Windows 3.1. To duplicate this, create a modal dialog with the styles CAPTION, MODAL FRAME, MINIMIZE-BOX, activate the dialog, press the Minimize button -- and watch it move to the top right corner, on top of the modal frame!

The workaround: don't use a Minimize box on a modal dialog.

Modifying common dialogs **93-01-20**

If you want to modify a common dialog, do not delete controls from the templates, as the common dialog procedures expect these controls to exist. Instead, make them invisible or move them outside the boundaries of the dialog gox to prevent the user from seeing or accessing them.

Null dialog handles from Borland custom dialogs **93-07-30**

If you keep getting null dialog handles with Borland C++ unless I have Turbo C++ or Borland C++ running, your dialog is probably of the BorDlg class, which requires code in BWCC.DLL. If you are running with the IDE active in Windows, the DLL is already loaded.

However, you have probably not done anything to force BWCC.DLL to be loaded with your application when running standalone, so the dialog manager cannot find the necessary routines to draw the dialog. The easiest way to force BWCC.DLL to be loaded is to call BWCCGetVersion() at the very beginning of your application, and to link in BWCC.LIB.

Alternately, you can do a

```
hBorlandDLL = LoadModule("loadbwcc.exe");
```

when you start up your application, remembering to release the handle before exiting your application..

Preventing switching away from a modal dialog

The design of the Windows API means that if there are two dialogs active simultaneously, the user can switch between the two, even one of them is modal. To prevent this, you should use EnableWindow() to explicitly diable any modeless dialogs when your modal dialog starts up.

Using a dialog as your main window **92-12-18**

First, you have to create a dialog box first. Include a class name(you name it). Then, in your WinMain function, register a class using that class name and add the constant DLGWINDOWEXTRA to the window extra byte component. This constant is defined in Windows.h. Then, use CreateDialog() to display the dialog, make sure the last 2 parameters are set to NULL. In your WndProc function, instead of calling DefWindowProc, call DefDlgProc. Then, you are basically all set. Make sure in your dialog definition you create your dialog box initially visible. For further information, see your docs.

Using Borland custom dialogs with other compilers 92-09-28

You can't integrate **bwcc.dll** with the Dialog Editor but you can manually modify the dialog file and use appropriate BWCC control classes and styles.

Include **bwcc.h** in your header file and then add **bwcc.lib** to your link options (*before* **libw.lib**). Also make sure **bwcc.dll** can be found in either the Windows directory or the current directory when the app starts or in the path.

Examples (thanks to Sam Espartero, *sqe@hpcc01.corp.hp.com*):

Link Options:

```
/align:16 /NOD PLAYCD USERCODE MCICDA SUPERCLS,PLAYCD.EXE,,
LIBW MLIBCEW bwcc mmsystem, PLAYCD.DEF
```

Dialog File:

```
ABOUT DIALOG   4, 5, 199, 137
STYLE WS_POPUPWINDOW | DS_MODALFRAME | WS_VISIBLE |
      WS_CLIPSIBLINGS | WS_DLGFRAME
CAPTION "Sample BWCC Dialog"
CLASS "BorDlg"
BEGIN
    CONTROL "", 100, "Static", SS_ICON | WS_CHILD | WS_VISIBLE,
            5, 16, 16, 16
    CONTROL "", -1, "BorShade", WS_CHILD | WS_BORDER,
            33,6,161,126
    ...

    CONTROL "", IDOK, "BorBtn", BS_DEFPUSHBUTTON | WS_TABSTOP |
            WS_CHILD, 154, 9, 32, 20
END
```

Message box:

```
#ifdef BWCC
    BWCCMessageBox(GetActiveWindow(), "Unknown MCI Error!",
                   "MCI Error", MB_OK | MB_ICONHAND);
#else
    MessageBox(GetActiveWindow(), "Unknown MCI Error!",
               "MCI Error", MB_OK | MB_ICONHAND);
#endif
```

Controls

Allowing ENTER in a multiline edit control **92-09-15**

To allow the use of the **Enter** key, there is no need to subclass the edit control. An easier way in Windows 3.1 and later (which also works better!) is to specify ES_WANTRETURN as part of the style for the edit control (see the Windows 3.1 SDK documentation for details).

Aligning multi-column listboxes **92-09-15**

In the resource file make sure the list box has the LBS_USETABSTOPS style. When you add the items to the listbox, separate the fields with tabs. You can either use the default tab stops, or set your own by sending the LBS_SETTABSTOPS message to the listbox. For more information, see the SDK Reference, volumes 1 and 2.

It is also possible to use a fixed font, but the tabstop solution usually ends up looking much better.

Changing button colors

In Windows 3.0, the button face is defined by two colors. The grey (white with EGA) face and a dark grey (grey if ega) shadow. The colors also change when the button goes from a normal to pushed in state. The WM_CTLCOLOR message only allows you to change one color at a time so to which of the button face colors should this apply? (Windows 2.x button faces had only one color so it made sense.)

Maybe something tricky could have been done by using the background color for the shadow and foreground color for the face and perhaps doing something strange to get the text color in another way... And how do you return 2 brushes (you now need a foreground and a background brush)? Or maybe even better, make colors a property of the window and some windows could have multiple color properties...

Anyway, Windows doesn't look at the WM_CTLCOLOR message for buttons and thus doesn't allow you to change the button colors. Try it with a listbox instead... The only way to change button colors is to specify *ButtonColor*=, *ButtonShadow*= and *ButtonText*= in the [Colors] section of your **win.ini** file.

In Windows 3.1, the button text, shadow and face colors can also be defined using the Control Panel.

Changing the font size in a dialog **94-02-22**

To change the font size of n individual control, send it a WM_SETFONT message. If you need to do it for all your controls in a dialog, enumerate the child windows and send the message in the enumeration callback function.

Combo boxes with tab stops **93-01-20**

Windows does not support combo boxes with tab stops. If you need such items, you have three choices:

- Use an owner-draw combo box. This will allow you to draw the items at the precise locations you need. Text-only owner-draw controls are not difficult to implement.
- Use GetTextExtent() to determine the length of each part, padding with blanks until it approximately lines up. Fairly easy, but not fast and not precise.
- Change the font of the control to either Courier or the fixed System font. While this is simple to do, your combo box font will not be consistent with the rest of your dialog box controls.

Controlling color in Borland dialogs **93-04-30**

Borland's custom dialog window classes do not pass through the usual WM_CTLCOLOR messages, thus making it impossible to set control colors. If you need to do this, you will need to create a non-Borland dialog, paint the embossed-look background yourself, and use Borland controls on this dialog.

Custom button bitmaps in Borland dialogs **92-11-15**

To create owner-drawn buttons for new additional buttons beyond those supplied in **bwcc.dll**, you first need to create the button bitmaps. Create them 63 pixels wide by 39 deep, and number them "logically".

The bitmap numbers are related to the resource id of the button that you want to use them. Use the following numbers:

button id + 1000	Unpressed bitmap
button id + 3000	Pressed bitmap, VGA and higher resolutions
button id + 5000	Focused bitmap, VGA and higher resolutions
button id + 2000	Unpressed bitmap, EGA
button id + 4000	Pressed bitmap, EGA
button id + 6000	Focused bitmap, EGA

If you use these bitmap ids, **bwcc.dll** will use the bitmaps on the buttons automatically.

Drawing on a dialog background **93-01-20**

To draw on a dialog background, you can either subclass the dialog, or, more simply, respond to the WM_PAINT message in your dialog procedure, calling ValidateRect() after you have redrawn the background.

Hiding dialog controls

```
EnableWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE), FALSE);
ShowWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE), SW_HIDE);
UpdateWindow(GetDlgItem(hDlg, IDD_CONTROLTOHIDE));
```

Listboxes with large amounts of data **92-11-15**

A standard listbox is limited to 32K of data. If you need a listbox with a larger number of items, you can use one of the following options:

- Use an owner-draw listbox with LBS_HASSTRINGS. This will allow 4096 items.
- Use an owner-draw listbox LBS_OWNERDRAWFIXED but without LBS_HASSTRINGS. This is somewhat more work, but will allow 8192 items.

Some third-party libraries also implement listbox classes which can handle huge amounts of data.

Finally, Microsoft has a virtual listbox implementation available by *ftp* at *ftp.uu.net*, in the directory */vendor/microsoft/developer-network*.

Subclassing standard controls

You can *subclass* standard controls by having your own window procedure handle the messages for the windows you create (using SetWindowLong()). The only caveat here is for useability: make sure that your subclassed controls don't behave in an unexpected manner.

What is definitely a bad idea is modify the *class* procedure of a standard control (using SetClassLong()) and changing the window procedure for *all* such windows, as this will affect all edit controls in all applications currently running in the Windows session.

Using a window as a modal dialog **92-12-15**

To use a window as a modal dialog, you will need to implement your own message loop for that window, complete with PeekMessage(), TranslateMessage() and DispatchMessage() calls. This will allow you to accept events only for the current window, discarding all other events.

Memory

Using new() in C++

In Borland C++ 2.0, and in 3.x's medium model, new() ends up calling LocalAlloc(), allocating memory from your near 64K segment. In BCC 3.x's large and compact models (and in Microsoft C/C++ 7.0), however, it will make one GlobalAlloc() and do subsegment allocations to allow you access to the full memory without making excessive demands on the system limit of 4096 (8192 in 386 enhanced mode) global memory handles.

Global memory owned by DLL

If you use GlobalAlloc in a DLL, the application that called the DLL will own the object. There is a way around this, though: allocate the memory using the GMEM_DDESHARE flag; this will make the allocating code segment (rather than the current task) own the memory.

Determining size of physical memory

You need to make a DPMI call to obtain that piece of information. DPMI call 0500h with ES:DI pointing to a 30h byte buffer returns the "Free Memory Information":

Offset	Description
00h	Largest available free block in bytes
04h	Maximum unlocked page allocation
08h	Maximum locked page allocation
0Ch	Linear address space size in pages
10h	Total number of unlocked pages
14h	Number of free pages
18h	Total number of physical pages
1Ch	Free linear address space in pages
20h	Size of paging file/partition in pages
24h-2Fh	Reserved

The size of one page in bytes can be determined by function 0604h, which returns the page size in bytes in BX:CX. To call a DPMI function, invoke the interrupt 31h. Carry bit will be clear if call was successful.

The complete DPMI 0.9 specification is available free from Intel Literature JP26, Santa Clara. It's also available on ftp.cica.indiana.edu.

GDI

Animation **92-10-06**

If you want to do good-quality animation under Windows 3.1 without requiring that each user have a 486/50 with an accelerated video card, you should consider using a differential animation technique. There is a good example available on *ftp.uunet.uu.net* (and also on CompuServe) under */vendor/microsoft/multimedia/sample* called **rleapp**, which uses this technique. Another sample program in the same directory, **transblt,** demonstrates a technique for doing fast BitBlits.

Both techniques are also documented in the technotes in */vendor/microsoft/multimedia/technote.*

Animation: WAP **93-04-25**

Another choice for animation is to use the Windows Animation Package (WAP) developed by Brian Goble *(goble@hardy.u.washington.edu);* information on the toolkit is available by email from Brian.

Background color

If you insist on a white background, use

```
WinClass.hbrBackground = GetStockObject(GCW_WHITEBRUSH);
```

for your window background. If it doesn't matter to you, however, you should use the *Control Panel*-defined window background color instead:

```
WinClass.hbrBackground = CreateSolidBrush(COLOR_WINDOW + 1);
```

Changing palette entries in 16-color mode

If you are using a standard driver, you will need to bypass Windows to do it (if you happen to have a 16-color driver which support palettes, you can use standard Windows palette management functions).

Microsoft will tell you to buy the DDK, but there is another way. Now, the Windows system palette maps onto the VGA 16-color palette as follows:

VGAPAL	SYSPAL	VGAPAL	SYSPAL
00	00	08	07
01	01	09	13
02	02	10	14
03	03	11	15
04	04	12	16
05	05	13	17
06	06	14	18
07	12	15	19

So you can define some macros to take care of the mapping:

```
#define syspal(n) (n<7 ? n : (n>8 ? n+4 : (n=7 ? 12 : 7)))
#define vgapal(n) (n<7 ? n : (n>12 ? n-4 : (n=7 ? 8 : 7)))
```

When you get a WM_SETFOCUS event, save the current state of the hardware colormap and installs the one you want. When you get a WM_KILLFOCUS event, restore the original palette. Don't use the palette registers directly, though, just modify the color registersthat they point to. (For details on redefining a VGA palette, see a book such as *A Programmer's Guide to PC and PS/2 Video Systems* by Richard Wilton.)

DIB bitmaps **92-10-07**

Microsoft has higher-level DIB library and DLL available for downloading from CompuServe. Unfortunately, it is not currently available by ftp.

Speeding up WM_PAINT redraws

To speed up your WM_PAINT processing, you may want to use a technique similar to the following one, as presented by John Grant *(jagrant@emr1.emr.ca):*

There are two reasons for redrawing your window:

- because Windows tells you to do it
- because something application specific requires it

You shouldn't have to redraw everything from scratch every time you get a WM_PAINT message - just save it as a bitmap and repaint from the bitmap. However, there are cases when the bitmap becomes invalid and you have to re-paint the hard way.

In the main window WndProc, I respond to Windows messages as follows (note there may be other code too, I just put in the stuff relevant to the redraw):

```
case WM_CREATE:      KillBitmap();
                     break;
case WM_SIZE:        KillBitmap();
                     break;
case WM_PAINT:       DrawMyPicture(hwnd);
                     break;
case WM_DESTROY:     KillBitmap();
                     PostQuitMessage(0);
                     break;
```

Notice that you should *not* do KillBitmap() when I handle WM_PAINT. If, in response to an application-specific condition, you want to force a redraw, do the following:

```
KillBitmap();
InvalidateRect(hwnd,NULL,TRUE);    // generates WM_PAINT
```

Now, for an application specific sample code fragment:

```
// global variable
    static HBITMAP hbitmap_main = NULL;

void DrawMyPicture(HWND hwnd)
{
    PAINTSTRUCT ps;
    HDC hdc;

    hdc = BeginPaint(hwnd, &ps);
    if (hbitmap_main == NULL){
        DrawMyPictureTheHardWay(hwnd, hdc);
        hbitmap_main = SaveClientAreaAsBitmap(hwnd);
    } else {
        DrawBitMap(hdc, hwnd, 0, 0, hbitmap_main);
    }
    EndPaint(hwnd,&ps);
}

void KillBitmap(void)
{
    if (hbitmap_main != NULL){
       DeleteObject(hbitmap_main);
```

```
        hbitmap_main = NULL;
    }
}
```

Don't use PAINTSTRUCT.rcPaint which describes the area that Windows says needs repainting; just BitBlt the whole thing and Windows will clip it. No, it's not overkill - it's fast!

Finally, two routines that you can put in your library for use with all your apps.

```
/*-------------------------------------------------------------------*
 | save entire client area of window into a bitmap                   |
 *-------------------------------------------------------------------*/

HBITMAP SaveClientAreaAsBitmap(HWND hwnd)
{
RECT rect;
HDC hdc,hmemdc;
HBITMAP hbitmap,old_hbitmap;

    hbitmap=NULL;
    hmemdc=NULL;
    hdc=NULL;

    //get source device context for the client area
    hdc=GetDC(hwnd);
    if(hdc==NULL) goto done;

    //get destination memory hdc compatible with client area
    hmemdc=CreateCompatibleDC(hdc);
    if(hmemdc==NULL) goto done;

    //create compatible bitmap for client area
    GetClientRect(hwnd,&rect);        //.top & .left are both 0
    hbitmap=CreateCompatibleBitmap(hdc,rect.right,rect.bottom);
    if(hbitmap==NULL) goto done;

    //select client area bitmap into device context so we can write on it
    old_hbitmap=SelectObject(hmemdc,hbitmap);

    //and copy it to the new hmemdc
    BitBlt( hmemdc,0,0,                 //destination (x,y)
            rect.right,rect.bottom, //width, height
            hdc,0,0,                    //source (x,y)
            SRCCOPY);

    //all done
    SelectObject(hmemdc,old_hbitmap);

done:    if(hdc    !=NULL) ReleaseDC(hwnd,hdc);
    if(hmemdc!=NULL) DeleteDC(hmemdc);

    return(hbitmap);
}
```

```
/*-------------------------------------------------------------*
 | Draw a bitmap into the current device context.               |
 | This is essentially the same as Petzold's code.             |
 *-------------------------------------------------------------*/
void DrawBitMap(HDC hdc,int xleft,int ytop,HBITMAP hbitmap)
{
RECT      rect;
HDC       hmemdc;
BITMAP    bm;
POINT     point;
HBITMAP old_hbitmap;

     hmemdc=NULL;
     old_hbitmap=NULL;

     //create memory device context & select bitmap
     hmemdc=CreateCompatibleDC(hdc);
     if(hmemdc==NULL) goto done;
     old_hbitmap=SelectObject(hmemdc,hbitmap);

     SetMapMode(hmemdc,GetMapMode(hdc));       //same as for hdc

     //get bitmap dimensions & convert to logical
     GetObject(hbitmap,sizeof(bm),&bm);
     point.x=bm.bmWidth;
     point.y=bm.bmHeight;
     DPtoLP(hdc,&point,1);

     BitBlt(hdc,xleft,ytop,                    //destination
               point.x,point.y,                //width, height
               hmemdc,0,0,                        //source
               SRCCOPY);

done:    if(old_hbitmap!=NULL) SelectObject(hmemdc,old_hbitmap);
     if(hmemdc!=NULL) DeleteDC(hmemdc);
     return;
}
```

Using CMY colors instead of RGB **93-01-20**

To use CMY colors with the Windows GDI instead of RGB, you can define the following macro to supplement the standard RGB one:

```
#define   CMY(c, m, y)       RGB(255 - c, 255 - m, 255 - y)
```

Using only solid colors **92-11-15**

If you want to use only solid colors (that is, get the nearest solid color to the one you specified), use the GetNearestColor() function call to map an RGB value to a solid cor available in the current color palette. Also, if you are using a palette, you can call the PALETTERGB(r,g,b) macro instead of the usual RGB(r,g,b) to map to the nearest color in the palette.

Text and fonts

Creating new fonts **93-01-20**

To create new TrueType fonts, you will need a commercial package such as Fontographer or FontMonger. While FontMonger is a more limited tool, it is quite inexpensive and often sufficient for smaller projects. Fontographer is a semi-professional tool with a higher price tag. Neither tool allows you to construct your own hints for small point sizes.

Determining font sizes **94-02-25**

Unfortunately Windows LOGFONT structure sets the font height using device units and not actual point sizes, and the common font dialog returns the selected size in the same units. This means that you need to do conversions similar to the ones shown below:

```
iDevRes = GetDeviceCaps( hDC, LOGPIXELSY );
iPointSize = MulDiv( iDeviceSize, 72, LOGPIXELSY );
```

or

```
iDevRes = GetDeviceCaps( hDC, LOGPIXELSY );
iDeviceSize = MulDiv( iPointSize, LOGPIXELSY, 72 );
```

Rotating fonts **93-01-20**

First, you cannot rotate screen fonts. Effectively this means that you can only rotate TrueType and Type 1 (ATM) fonts. To do the rotation, you will have to create a new logical font with the correct escapement value. Select the font into the display context, do your displaying, select the original font back in, and delete the font object.

Note that Type 1 fonts, which are considered device fonts, use a reversed direction for the rotation from TrueType fonts; you'll have to check the font type before doing the rotation, or otherwise you will have some fonts angled up and others down.

TrueType width calculation **92-11-03**

The most accurate widths for TrueType fonts can be obtained using the following method (courtesy of Glenn Adams *(glenn@wheat-chex.ai.mit.edu)):*

1. Using EnumFonts() (or EnumFontFamilies()), obtain lpntm->ntmSizeEM. This value "specifies the size of the em square for the font, in the units for which the font was designed (notional units)." Most fonts will have the value of 2048.
2. Create a font using the above size as the lfHeight. This will create a font with metrics that coincide with the coordinate space used in the font's design, i.e., no scaling will occur in the logical coordinate space.
3. Use GetCharABCWidths() to get the ABC width structures for all the font's glyphs. You can now compute the widths quite accurately, along with overhang and overhang. Keep in mind that the horizontal escapement (i.e., the distance current point is advanced after rendering a glyph) is equal to A + B + C of the returned widths. If A is negative, the glyph extends outside the EM square to the left; if C is negative, then the glyph extends outside the EM square to the right.

One last point: the above widths will not take into account grid fitting, which will occur in actual display. But, if you set the mapping mode to MM_ISOTROPIC and use a logical coordinate space which coresponds to the font design size, you can do all your layout computations using design sizes without regard to final viewport mapping.

Kernel and low-level programming

Activating the previous instance **93-01-20**

If you want to run only a single instance of your application, and activate the first instance when the user tries to run a second one, use the following code at the beginning of your WinMain function:

```
if ( hPrevInstance ) {
    BringWindowToTop( FindWindow( "MyClassName", NULL ));
    return( FALSE );
}
```

Note that the value of hPrevInstance will always be NULL on Windows NT.

VxD development **94-02-25**

To develop VxDs, you will need to either purchase the Windows DDK (Device Driver Kit) or subscribe to the Microsoft Developer Network CD-ROM Level 2, which includes all of Microsofts SDKs and DDKs, as well as a plethora of developer information. If you own a CD-ROM drive, you should likely purchase the MSDN/L2, as the cost of the DDK alone is almost the same as that of the CD-ROM subscription.

VxD technical notes and samples **94-02-25**

The following technical notes are available by ftp from *ftp.microsoft.com:*

Q29519.txt	10-92.zip	Writing an Installable Driver technical article
Q97373.txt	10-50.zip	The VxD-Lite Mini-DDK technical article
Q97323.txt	7-9.zip	VIdleD: A VxD that Demonstrates the Call_When_Idle service
Q97322.txt	7-8.zip	VHook86D: A VxD that Hooks Interrupt 2Fh in V86 Mode
Q97321.txt	7-7.zip	VFIntD: A VxD that Captures Floppy Disk Interrupts
Q97320.txt	7-6.zip	VDialog: A VxD that Demonstrates How to Serialize I/O
Q97319.txt	7-3.zip	GPTrap: A VxD that Traps GP Faults
Q97318.txt	7-2.zip	Generic: Illustrates the Basic Structure of a VxD
Q97317.txt	7-19.zip	VxD Tools: Used for Building Virtual Devices
Q97316.txt	7-18.zip	VxD Include Files: Used for Building Virtual Devices
Q97315.txt	7-17.zip	VWFD: A VxD that Reports a VM's Windowed vs. Full-Screen State
Q97314.txt	7-16.zip	VWatchD: Illustrates the Basic Structure of a VxD
Q97313.txt	7-15.zip	VNMID: A VxD that Hooks the Non-Maskable Interrupt
Q97312.txt	7-14.zip	VMPages: A VxD that Exports a VMM Service to an Application
Q97311.txt	7-13.zip	VMIRQD: A VxD that Virtualizes a Hardware Interrupt
Q97310.txt	7-12.zip	VMIOD: A VxD that Monitors I/O Traffic on a Port
Q97309.txt	7-11.zip	VKXD: A VxD that Simulates the ALT+ENTER Key Combination
Q97308.txt	7-10.zip	VITD: A VxD that Simulates an Interval Timer Device
Q97307.txt	7-1.zip	Eatpages: A VxD that Consumes Physical Pages

VFOOD.ZIP	A Basic Windows Virtual Device
RING0.ZIP	MSJ Source: May 1993
VXD.ZIP	MSJ Source: October 1992
VXD1.ZIP	MSJ Source: February 1993

The file locations are the following:

Q.txt*	/devtools/winsdk/kb
7-.zip:*	/msdn/vxdsamp
10-.zip:*	/msdn/vxdsamp
.zip	/softlib/mslfiles

VxD developer documentation **94-02-25**

The development of VxDs (virtual device drivers) is closely related to device driver development. The following is a partial list of VxD development resources:

- *Microsoft Developers Network CD.* Microsoft Corp.
- Norton, Daniel A.: *Writing Windows Device Drivers.* Addison-Wesley. ISBN 0-201-57795-X
- Pietrek, Matt: *Microsoft Systems Journal.* Run Privileged code from Your Windows-based Program Using Call Gates. May 1993.
- Salter, Bret: *Dr. Dobbs Journal.* An Exception Handler for Windows 3. September 1992.
- Schulman, Andrew: *Microsoft Systems Journal.* Go Anywhere, Do Anything with 32-bit Virtual Device Drivers for Windows. October 1992.
- Schulman, Andrew: *Microsoft Systems Journal.* Exploring Demand-Paged Virtual Memory in Windows Enhanced Mode. December 1992.
- Schulman, Andrew: *Microsoft Systems Journal.* Call VxD Functions and VMM Services Easily Using Our Generic VxD. February 1993.
- Smith, Gordon: *Microsoft Systems Journal.* Embedded Device Drivers simplify the support of unusual devices under Windows. May 1991.
- Thielen, David: *Windows Tech Journal.* Various articles, April 1992 to August 1992.
- Thielen, David and Bryan Woodruff: *Writing Windows Virtual Device Drivers.* Addison-Wesley, 1993, $43.95. ISBN 0-201-62706-X.

Getting the instance handle **92-09-28**

In general, it's usually best to store the instance handle in a global variable, so that it will always be available. If, however, you don't have it handy, you can get it with an easy Windows API call:

```
GetWindowWord(hWnd, GWW_HINSTANCE)
```

I/O ports and Windows **94-02-25**

Windows generally works fine with I/O commands (putc, getc, putw and getw), and problems with I/O usually involve card configuration problems.

- Take care to avoid electro static discharge damage.

- Check you card addresses carefully. Some cards use switches that are "1" or "on" in the off or down position. Remember that card addresses are usually set in hexadecimal. Avoid addresses less than 0x100, as your system my have some devices down there.

- Most cards reserve multiple output ports; make sure your card does not conflict with other cards or the system. Typically cards reserve a power of 2 ports: 2, 4, 8, 16, 32 or more. Align card adresses to multiples of the required ports. A 16 decimal port card will have and address like 0x*nn*0. Don't overlap a card even if it uses less than a power of 2 ports. For example if a card reserves 12 decimal ports at address 0x110 then the next available port is 0x120. Don't try to sneak a 4 port card in at 0x11C.

- Make sure the other switches/jumpers on your card are configured properly.

- If necessary remove other cards to trouble-shoot. A hardware problem with one card can interfere with other cards.

Thanks to Larry Kucera.

Restarting Windows **92-10-05**

To restart Windows, use:

```
ExitWindows(EW_RESTARTWINDOWS, 0)
```

Rebooting the system **92-10-05**

To reboot the entire system, use:

```
ExitWindows(EW_REBOOTSYSTEM, 0)
```

OLE and DDE

Applying OLE technology **94-03-10**

The Microsoft Internet anonymous ftp server ftp.microsoft.com now contains *OLE-info.** (in RTF, EPS, Word 2 and Word 6 formats). This is a 65 page document called "How To Apply OLE 2 Technology in Applications", which has two sections.

The first section lists various application categories, like database, DTP, spreadsheet, drawing/CAD, etc., and how applications that call into such categories might use each OLE 2 technology (component objects, compound files, data transfer Drag & Drop, automation, and compound documents). Some applications, of course, are not suitable for using all technologies, so this section helps you decide what is really important for your application.

The second section contains all the same information, but organized instead by OLE 2 technology, explaining how each application category would use that specific technology. This is useful for evaluating the full value of a given technology in OLE 2 for multiple applications, and gives you a few ideas on what sorts of interesting things you might do with the technology.

OLE resources on ftp.microsoft.com **94-03-19**

The following list contains a brief description of the OLE 2 resources available from *ftp.microsoft.com*.

- */drg/ole-info/inside-ole2-code-update*
  An update to the source code from *Inside OLE 2* by Kraig Brockschmidt.
- */drg/ole-info/applying-ole*
  A guide to applying OLE 2 technology.
- */drg/ole-info/background/decolepr.doc*
  Microsoft-DEC Common Object Model agreement.
- */drg/ole-info/background/mfc25.doc*
  A backgrounder on MFC 2.5s OLE 2 classes.
- */drg/ole-info/background/olebg.doc*
  OLE 1 background, mostly of historical interest.
- */drg/ole-info/background/olecom.doc*
  Distributed OLE backgrounder.
- */drg/ole-info/background/ olemngbg.doc*
  OLE 2 backgrounder for management and non-technical staff.
- */drg/ole-info/background/oletech.doc*
  A technical OLE backgrounder with more technical info and less hype.
- */drg/ole-info/background/olevisbg.doc*
  Common Object Model vision backgrounder.
- */drg/ole-info/ole201*
  OLE 2.01 documentation, including the Design Specification.

Using NetDDE **92-12-20**

To use an existing DDE application over the network you simply need to create a DDE share for it with a separate utility. The DDE share you create refers to a specific Service|Topic pair, and when a client connects to that share, it is as if they had connected to the underlying Service|Topic. (The DDEShare tool that comes with the Windows for Workgroups resource kit will do this.) From the client application you simply connect to \\MachineName\NDDE$|DDEShareName instead of Service|Topic.

Thanks to Blake Coverett of Microsoft Canada for this information.

Miscellaneous

Accessing C++ classes in a DLL **93-03-40**

It's definitely possible to access C++ classes which are implemented in a DLL. The following methods are possible:

- Use extern "C" function definitions, and manually create a **.def** file for the C-defined functions in the DLL, and then use ImpLib (in the Windows SDK) to create a **.lib** file for the DLL.
- Use standard C++ function definitions, and manually create a **.def** file for the mangled C++ function names in the DLL, and then use ImpLib to create a **.lib** file for the DLL.

Changing your current directory

The easy way is to use DlgDirList(). You can specify zero for the two ID fields. You can use the current window ID for the dialog handle field.

The standard C library functions chdir() and getcwd() can also be used.

Detecting idle time **93-05-08**

The 'idle-detecting' message loop may suit your case. That is, replace the standard while(GetMessage()) ... in WinMain() with the something like the following (thanks to Risto Lankinen for the example and to Raymond C. at Microsoft for the warning note):

```
if ( PeekMessage(&msg) ) {
    // The queue contains messages - process them
    // GetMessage() would automatically detect WM_QUIT, but
    // we must explicitly check for it.
    if ( msg.message == WM_QUIT )
        break;
    if ( TranslateAccelerator(hWnd,hAcc,&msg) )
        continue;

    TranslateMessage( &msg );
    DispatchMessage( &msg );

    // You might want to save the last time a message
    // was processed
    dwLastMsgTime = GetTickCount();
} else {
    // The queue is empty - user is doing nothing with *this* app
    if ( GetTickCount() < dwLastMsgTime ) {
        // Timer wrapped around -- do something!
    } else if ( GetTickCount() - dwLastMsgTime > MSGTIMEDELTA ) {
        // Do something funky
    }
```

Note that since the application is now PeekMessage-based, Windows itself will never go idle. (This is explicitly mentioned in the SDK under PeekMessage.) This has at least two consequences.

- In enhanced mode, DOS applications will not receive as much CPU as they might otherwise, since some of the CPU is being spent by Windows.

- If you are running a laptop computer, the automatic power-save function will never kick in.

Enumerating active processes **92-12-15**

To enumerate the processes currently active in the system (to build a process tree, create a task manager or otherwise), you should use TaskFirst(), TaskNext() and IsTask(). All of these are included in **toolhelp.dll**, a redistributable component of the Windows 3.1 SDK..

Extracting icons from a .EXE or .DLL

In Windows 3.1, it's easy to enumerate the icons in a Windows EXE or DLL even if you don't already know their names. SHELL.DLL exports

HICON ExtractIcon(hInst, lpszExeName, nIcon)

This function returns a handle to the specified icon (where 0 is the default icon displayed by Program Manager), or the number of icons in the file if you pass in an index of -1.

Better yet, SHELL.DLL also exports the function:

FindExecutable(lpszFile, lpszDir, lpszResult)

which will give you the executable associated with a given document file. You can then extract the appropriate icon from that file.

Finding the directory: application program **92-09-28**

To locate your executable program directory , use:

```
GetModuleFileName(hInstance, (LPSTR) szPath, sizeof(szPath));
```

Finding the directory: initial **93-04-30**

To locate your initial directory , use:

```
_getcwd(szPath, sizeof(szPath));
```

In Visual Basic, you can use the CurDir$ variable.

Finding the directory: system **93-04-30**

To locate your Windows system directory , use:

```
_getcwd(szPath, sizeof(szPath));
GetWindowsDir(szPath, szWinPath, sizeof(szWinPath));
GetSystemDir(szWinPath, szSysPath, sizeof(szSysPath)));
```

Finding the directory: Windows **93-04-30**

To locate your Windows directory , use:

```
_getcwd(szPath, sizeof(szPath));
GetWindowsDir(szPath, szWinPath, sizeof(szWinPath));
```

Finding the number of instances running

Use the following code:

```
nNumInsts = GetModuleUsage(hInstance);
```

Note that this will always return 1 within Windows NT.

Multimedia RIFF file format **92-09-15**

The IBM/Microsoft RIFF and MCI definition document is available for anonymous ftp download from the */vendor/microsoft/multimedia* directory on *ftp.uu.net*. If you don't have anonymous ftp access, try CompuServe in the WINSDK forum, or Microsoft Online.

The document describes multimedia interfaces (MCI) and data formats (RIFF). IBM has committed to include support for these in OS/2. These interfaces are already supported in the Multimedia Extensions for Windows (MME) and Windows 3.1. Included in the RIFF file format is a waveform (.WAV) audio definition; this format is the system standard for Windows and OS/2.

Using CARDS.DLL for your own games **93-04-30**

The following include files document the interface to **cards.dll**, allowing you to access the card functionality in your own games. Note, however, that **cards.dll** is copyrighted by Microsoft and can't be freely distributed. However, if all your users have a Windows Entertainment Pack, you're all set!

Thanks to Heath Ian Hunnicutt (heathh@cco.caltech.edu) for disassembling CARDS.DLL and coming up with the basis for this documentation.

- **cards.h**

```
#include "crd.h"

/*
 *  cdtInit: Initializes the cards module
 *
 *  pdxCard: receives the width of the card
 *  pdyCard: receives the height of the card
 *  returns: success or failure
 */
BOOL FAR PASCAL cdtInit(int FAR *pdxCard, int FAR *pdyCard);

/*
 *  cdtDrawExt: Draw a card with specified extents (size)
 *
 *  x, y:      location to draw the card
 *  dx,dy:     size to be drawn
 *  cd:        card to be drawn
 *  mode:      transfer mode
 *  returns: success or failure
 */
BOOL FAR PASCAL cdtDrawExt(HDC hdc, int x, int y, int dx, int dy,
                                    int cd, int mode, DWORD rgbBgnd);

/*
 *  cdtDraw: Draw a card
 *
 *  x, y:      location to draw the card
 *  cd:        card to be drawn
 *  mode:      transfer mode
 *  returns: success or failure
 */
BOOL FAR PASCAL cdtDraw(HDC hdc, int x, int y, int cd, int mode,
                                  DWORD rgbBgnd);

/*
 *  cdtTerm: terminates and cleans up the cards module
 */
void FAR PASCAL cdtTerm();
```

- **cdt.h**

```
#define CLOADMAX        5

/* Command ids */
#define IDACLUBS          1
#define ID2CLUBS          2
```

```
#define ID3CLUBS        3
#define ID4CLUBS        4
#define ID5CLUBS        5
#define ID6CLUBS        6
#define ID7CLUBS        7
#define ID8CLUBS        8
#define ID9CLUBS        9
#define IDTCLUBS        10
#define IDJCLUBS 11
#define IDQCLUBS        12
#define IDKCLUBS        13

#define IDADIAMONDS     14
#define ID2DIAMONDS     15
#define ID3DIAMONDS     16
#define ID4DIAMONDS     17
#define ID5DIAMONDS     18
#define ID6DIAMONDS     19
#define ID7DIAMONDS     20
#define ID8DIAMONDS     21
#define ID9DIAMONDS     22
#define IDTDIAMONDS     23
#define IDJDIAMONDS     24
#define IDQDIAMONDS     25
#define IDKDIAMONDS     26

#define IDAHEARTS       27
#define ID2HEARTS       28
#define ID3HEARTS       29
#define ID4HEARTS       30
#define ID5HEARTS       31
#define ID6HEARTS       32
#define ID7HEARTS       33
#define ID8HEARTS       34
#define ID9HEARTS       35
#define IDTHEARTS       36
#define IDJHEARTS       37
#define IDQHEARTS       38
#define IDKHEARTS       39

#define IDASPADES       40
#define ID2SPADES       41
#define ID3SPADES       42
#define ID4SPADES       43
#define ID5SPADES       44
#define ID6SPADES       45
#define ID7SPADES       46
#define ID8SPADES       47
#define ID9SPADES       48
#define IDTSPADES       49
#define IDJSPADES       50
#define IDQSPADES       51
#define IDKSPADES       52

#define IDGHOST53
```

```
#define IDFACEDOWN1   54
#define IDFACEDOWN2   55
#define IDFACEDOWN3   56
#define IDFACEDOWN4   57
#define IDFACEDOWN5   58
#define IDFACEDOWN6   59
#define IDFACEDOWN7   60
#define IDFACEDOWN8   61
#define IDFACEDOWN9   62
#define IDFACEDOWN10 63
#define IDFACEDOWN11 64
#define IDFACEDOWN12 65

#define IDFACEDOWNFIRST IDFACEDOWN1
#define IDFACEDOWNLAST IDFACEDOWN12

#define IDX 67
#define IDO 68
#define IDMAX IDDECK

/* internal IDs for animation */
#define IDASLIME1 678
#define IDASLIME2 679
#define IDAKASTL1 680
#define IDAFLIPE1 681
#define IDAFLIPE2 682
#define IDABROBOT1 683
#define IDABROBOT2 684

/* Red non-face card frame */
#define IDFRAME 999

#define FACEUP            0
#define FACEDOWN          1    /* for compatibility with old apps, use
                                  IDFACEDOWNFIRST..IDFACEDOWNLAST */
#define HILITE            2
#define GHOST             3
#define REMOVE            4
#define INVISIBLEGHOST 5
#define DECKX             6
#define DECKO             7
```

- **crd.h**

```
#include "cdt.h"

typedef int CD;

// CaRD struct, this is what a card be
typedef struct _crd {
        unsigned cd   : 15;                  // card # (1..52)
        unsigned fUp : 1;           // is this card up/down
        PT pt;                               // upper-left corner of card
        } CRD;
```

```
/* WARNING: Order of su's is assumed */
#define suClub 0
#define suDiamond 1
#define suHeart 2
#define suSpade 3
#define suMax 4
#define suFirst suClub

#define raAce 0
#define raDeuce 1
#define raTres 2
#define raFour 3
#define raFive 4
#define raSix 5
#define raSeven 6
#define raEight 7
#define raNine 8
#define raTen 9
#define raJack 10
#define raQueen 11
#define raKing 12
#define raMax 13
#define raNil 15
#define raFirst raAce

typedef int RA;
typedef int SU;

#define cdNil 0x3c

#define cIDFACEDOWN (IDFACEDOWNLAST-IDFACEDOWNFIRST+1)

#define SuFromCd(cd) ((cd)&0x03)
#define RaFromCd(cd) ((cd)>>2)
#define Cd(ra, su) (((ra)<<2)|(su))
```

Waiting for completion of WinExec() **93-01-20**

If you need to wait for the completion of a WinExec()'d process, Microsoft's recommended approach is to use the functions in **toolhelp.dll** (a free redistributable part of the Windows 3.1 SDK) to register a notification callback routine, which will then get called once the WinExec()'d application terminates.

Wsprintf and sprintf

wsprintf() can not print floating-point numbers by design. To print floating-point, you must use sprintf(). Remember, though, that all strings passed to wsprintf() should be cast to FAR!

Development tool specific issues

Borland ObjectWindows Library

A dialog as an MDI child window [Borland OWL] 93-01-20

If you want to use a dialog as an MDI child window, you can use the following code as the basis of your own implementation. Thanks to Martin Calsyn *(martin@iastate.edu)* for the sample code.

What follows are three files:

TMDIDLOG.H	The interface for modeless MDI-child dialogs.
TMDIDLOG.CPP	The implementation of same
EXAMPLE.CPP	An (partial) example of using the TMDIDLOG classes.

You need to sub-class TMDIDialogWindow in order to give your MDI child a unique icon, etc (not shown in the example) and to implement the CreateChildDialog abstract method. And, of course, you need to sub-class TMDIDialog in order to establish a transfer structure, interface objects and all the things you usually do with sub-classes of TDialog.

The classes included below work by presenting the dialog as a child of a TBWindow which is itself a child of the MDI. These classes are based on tips and suggestions offered by the Borland advisor line.

Martin claims that the transfer of this technique to MS MFC is trivial.

---------- TMDIDLOG.H ----------

```
#ifndef __tmdidlog_h__
#define __tmdidlog_h__

#include <owl.h>
#include <bwindow.h>

_CLASSDEF(TMDIDialog)
_CLASSDEF(TMDIDialogWindow)

class TMDIDialog : public TDialog
{
public:
    TMDIDialog(PTWindowsObject AParent,LPSTR AName,PTModule AModule=NULL) :
    TDialog(AParent, AName, AModule) {};
    TMDIDialog(PTWindowsObject AParent,int ResourceId,PTModule AModule=NULL):
    TDialog( AParent, ResourceId, AModule ) {};
    virtual void WMSetFocus( RTMessage Msg ) = [WM_FIRST + WM_SETFOCUS];
    virtual void CloseWindow( int ret );
    virtual void CloseWindow();
    virtual void Cancel( RTMessage msg ) = [ID_FIRST + IDCANCEL];

    friend class TMDIDialogWindow;

protected:
    TMDIDialogWindow   *mdiw;
};

class TMDIDialogWindow : public TBWindow
{
public:
    TMDIDialogWindow(PTWindowsObject aParent,LPSTR aName,
```

```
                        PTModule aModule = NULL);
    virtual void WMMDIActivate( RTMessage Msg ) = [WM_FIRST + WM_MDIACTIVATE];
    virtual void WMSetFocus( RTMessage Msg ) = [WM_FIRST + WM_SETFOCUS];
    virtual void WMSize( RTMessage Msg ) = [WM_FIRST + WM_SIZE];
    virtual void SetupWindow();
    virtual LPSTR GetClassName() { return "_TMDIDialogWindow"; };
    virtual void GetWindowClass( WNDCLASS& aWndClass );
    virtual void WMGetMinMaxInfo( RTMessage msg ) = [WM_FIRST +
WM_GETMINMAXINFO];
    virtual TMDIDialog *CreateChildDialog() = 0;

private:
    PTMDIDialog dlog;
};

#endif

------------------------ TMDIDLOG.CPP -------------------------

#include "tmdidlog.h"

TMDIDialogWindow::TMDIDialogWindow(PTWindowsObject aParent,
                        LPSTR aName,
                        PTModule aModule) :
          TBWindow(aParent,aName,aModule)
{
    dlog = NULL;
    GetApplication()->SetKBHandler( this );
    SetFlags( WB_KBHANDLER, FALSE );
}

void TMDIDialogWindow::WMMDIActivate( RTMessage Msg )
{
    TWindow::WMMDIActivate( Msg );

    if ( Msg.WParam == TRUE ) {
        GetApplication()->SetKBHandler(this);
        if (dlog!=NULL)
            SetFocus(((PTDialog)dlog)->HWindow );
    }
}

void TMDIDialogWindow::GetWindowClass( WNDCLASS& aWndClass )
{
    TWindow::GetWindowClass(aWndClass);
}

void TMDIDialogWindow::WMSetFocus(RTMessage Msg)
{
    PostMessage(HWindow, WM_MDIACTIVATE, TRUE, 0L );
    DefWndProc(Msg);
}

void TMDIDialogWindow::SetupWindow()
{
```

```
    TWindow::SetupWindow();

    dlog = CreateChildDialog();
    if (dlog) {
        dlog->mdiw = this;
        RECT r;
        GetApplication()->MakeWindow(dlog);
        SetFocus(dlog->HWindow);
        GetWindowRect(HWindow,&r);
        Attr.X = r.left;
        Attr.Y = r.top;
        GetWindowRect(dlog->HWindow,&r);
        AdjustWindowRect(&r,WS_CAPTION|WS_BORDER,0);
        Attr.W = r.right - r.left;
        Attr.H = r.bottom - r.top;
        MoveWindow(HWindow,Attr.X, Attr.Y, Attr.W, Attr.H, 1);
    }
}

void TMDIDialogWindow::WMGetMinMaxInfo( RTMessage msg )
{
    if (dlog) {
        MINMAXINFO *lpmmi = (MINMAXINFO *)msg.LParam;
        RECT r;
        POINT pt1,pt2;
        GetWindowRect(HWindow,&r);
        pt1.x = r.left;
        pt1.y = r.top;
        GetWindowRect(dlog->HWindow,&r);
        AdjustWindowRect(&r,WS_CAPTION|WS_BORDER,0);
        pt2.x = r.right - r.left;
        pt2.y = r.bottom - r.top;
        lpmmi->ptMaxSize = pt2;
        lpmmi->ptMaxPosition = pt1;
        lpmmi->ptMinTrackSize = pt2;
        lpmmi->ptMaxTrackSize = pt2;
    }
}

void TMDIDialogWindow::WMSize( RTMessage Msg )
{
    RECT r;
    TWindow::WMSize( Msg );
    if (!IsIconic(HWindow)) {
        GetWindowRect(dlog->HWindow,&r);
        AdjustWindowRect(&r,WS_CAPTION|WS_BORDER,0);
        Attr.W = r.right - r.left;
        Attr.H = r.bottom - r.top;
        MoveWindow(HWindow, Attr.X, Attr.Y, Attr.W, Attr.H, 1);
    }
}

//
//  TMDIDialog methods...
//
```

```
void TMDIDialog::WMSetFocus( RTMessage Msg )
{
    GetApplication()->SetKBHandler( this );
    DefWndProc( Msg );
}

void TMDIDialog::CloseWindow(int ret)
{
    TDialog::CloseWindow(ret);
    mdiw->CloseWindow();
}

void TMDIDialog::CloseWindow()
{
    TDialog::CloseWindow();
    mdiw->CloseWindow();
}

void TMDIDialog::Cancel(RTMessage msg)
{
    CloseWindow(IDCANCEL);
}

--------------------------- EXAMPLE.CPP --------------------------

//
//  #include owl, tmdidlog.h and others as needed
//

//
//  ChildDialog is a dialog box defined in the RC file under
//    the name 'MY_DLOG'.   The definition of MY_DLOG is pretty plain
//    except that it should have the following attributes:
//                  STYLE WS_CHILD | WS_VISIBLE
//    and optionally CLASS "BORDLG"
//  That is, it should have no caption, sys menu, or border of any
//  type.   It should also have an origin of 0,0 otherwise it will
//  be offset within the MDI child window.
//
//  An instance of ClientDialog can be created as a modeless MDI
//  client by doing : `new ChildDialogWindow(this, "Title", NULL)'
//  from your MDI frame object or other appropriate place.
//

class ChildDialog : public TMDIDialog
{
public:
    ChildDialog(PTWindowsObject aParent,
                    LPSTR aName, PTModule aModule = NULL);

};

ChildDialogWindow::ChildDialogWindow(PTWindowsObject aParent,LPSTR aName,
                                        PTModule aModule)
```

```
                : TMDIDialogWindow(aParent,aName,aModule)
{
    obj_id = object_id;
}

TMDIDialog* ChildDialogWindow::CreateChildDialog()
{
    return (TMDIDialog*)new ChildDialog(this,(LPSTR)"MY_DLOG",NULL);
}

ChildDialog::ChildDialog(PTWindowsObject aParent,
                                    LPSTR aName, PTModule aModule)
                : TMDIDialog(aParent,aName,aModule)
{
    // The usual stuff:   Initialize the transfer structure, create
    //   interface objects, etc.

}
```

Microsoft Foundation Classes

Disabled menu choices become enabled **93-04-25**

By default, a disabled menu choice become enabled as soon as you add an ON_COMMAND handler for it. To make the menu item stay disabled, you will need to disable it in the ON_UPDATE_COMMAND_UI handler.

Listbox contents not available after dialog **94-02-25**

Using a dialog class's listbox member similar to the following

```
if ( DoDialog() == IDOK )
        iChoice = mListBox.GetSelection();
```

will not work. Even though the class members exist after the DoDialog, the dialog *window* has been destroyed, and its contents (the controls) with it.

Maximizing the initial window **94-02-25**

In the "InitInstance" code, where you create the main window, add code something like this:

```
CMainFrame* pMainFrame = new CMainFrame;
if ( !pMainFrame->LoadFrame( IDR_MAINFRAME, dwStyle ))
        return FALSE;
pMainFrame->ShowWindow( SW_SHOWMAXIMIZED );
pMainFrame->UpdateWindow();
```

Sluggish menus when menu prompts missing **93-07-15**

If the menus in your MFC2 application appear very sluggish whenever there are one or more menu items without prompts, there is nothing to worry about. MFC is simply attempting to write debugging messages on the debugging device (AUX). These messages do not appear (and thus the menus are not sluggish) if the code is complied with the *release* option instead of *debug.*

You can use a driver for a second monitor, connect a serial terminal to COM2, or run DebugWin to catch the warning messages.

Using CTL3D with MFC **93-04-30**

The current version of **ctl3d.dll** is not supported for use with the Microsoft Foundation Classes; Microsoft is working on an update that will be fully compatible. However, you can use the current version, too, with no apparent problems (except the inability to use a message map outside of a dialog box). While dialogs automatically receive the 3-D effect, you can use the following technique outside of dialogs:

- Create a new class which inherits from the logical control class. (For example, class C3dStatic : public CStatic).

- Overload create() to do the following.
  1. Get a HWND using ::CreateWindow("Static"...
  2. Do the stuff desribed in the ctl3d help file section about using non-dialog controls without MFC
  3. do a CWnd::Attach(HWND)

- Remeber to do a detach() in the ~C3dStatic

Turbo Pascal for Windows

Using CTL3D.DLL with Turbo Pascal **93-01-20**

To use the **ctl3d.dll** (which provides a standardized 3-D look to all your dialogs, similar to Microsoft Excel 4.0) with Turbo Pascal, you'll need a unit file. The unit file and example below were contributed by Andreas Furrer (s_furrer@ira.uka.de).

CTL3D.PAS

```
(************************************************)
(*    Unit CTL3D                                *)
(*                                              *)
(*    for use with CTL3D.DLL from Microsoft     *)
(************************************************)

unit Ctl3D;

interface

uses WinTypes;

const CTL3D_BUTTONS        = $0001;
      CTL3D_LISTBOXES      = $0002;
      CTL3D_EDITS          = $0004;
      CTL3D_COMBOS         = $0008;
      CTL3D_STATICTEXTS    = $0010;
      CTL3D_STATICFRAMES = $0020;
      CTL3D_ALL            = $ffff;

function Ctl3dGetVer : WORD;
function Ctl3dSubclassDlg(HWindow : HWND; Ctrls : WORD) : bool;
function Ctl3dSubclassCtl(HWindow : HWND) : bool;
function Ctl3dCtlColor(DC : HDC; Color : TColorRef) : HBrush;          {ARCHAIC,
use Ctl3dCtlColorEx}
function Ctl3dEnabled : bool;
function Ctl3dColorChange : bool;
function Ctl3dRegister(Instance : THandle) : bool;
function Ctl3dUnregister(Instance : THandle) : bool;
function Ctl3dAutoSubclass(Instance : THandle) : bool;
function Ctl3dCtlColorEx(Message, wParam : WORD; lParam : LONGINT)
                                : HBrush;


implementation

function Ctl3dGetVer;          external 'Ctl3d' index 1;
function Ctl3dSubclassDlg;   external 'Ctl3d' index 2;
function Ctl3dSubclassCtl;   external 'Ctl3d' index 3;
function Ctl3dCtlColor;       external 'Ctl3d' index 4;
function Ctl3dEnabled;         external 'Ctl3d' index 5;
function Ctl3dColorChange;   external 'Ctl3d' index 6;
function Ctl3dRegister;       external 'Ctl3d' index 12;
function Ctl3dUnregister;     external 'Ctl3d' index 13;
function Ctl3dAutoSubclass; external 'Ctl3d' index 16;
function Ctl3dCtlColorEx;     external 'Ctl3d' index 18;
end.
```

Test3D.PAS

```
(************************************************)
(*    Test3D                                    *)
(*                                              *)
(*    Copyright (c) 1993 by A. Furrer           *)
(************************************************)

program Test3D;

{$R test3d.res}

uses WinTypes, WinProcs, WObjects, Ctl3d;

type
    Applikation =
        object(TApplication)
            procedure InitMainWindow; virtual;
        end;

    PMainwindow = ^TMainwindow;
    TMainwindow =
        object(TWindow)
            procedure WMLButtonDown(var Msg : TMessage); virtual
                                    wm_first + wm_LButtonDown;
        end;

procedure TMainwindow.WMLButtonDown;
begin
  Application^.ExecDialog(NEW(PDialog,Init(@Self,'Test3D')));
end;

procedure Applikation.InitMainWindow;
begin
  MainWindow := New(PMainwindow, Init(nil, 'Test3D'));
end;

var Prg : Applikation;

begin
    Ctl3dRegister(HInstance);
    Ctl3dAutoSubclass(HInstance);

    Prg.Init('Test3D');
    Prg.Run;
    Prg.Done;

    Ctl3dUnregister(HInstance);
end.
```

Visual Basic

Accessing I/O ports **94-02-25**

In order to access I/O ports from Visual Basic, download the file **inpout.zip** from the ftp site *ftp.cica.indiana.edu.*

Allowing user interaction in tight loops **92-12-20**

If your application uses a tight loop to perform calculations (or do ther things), you should call DoEvents() from within your loop to allow the user to activate other applications or to press the *Cancel* button in your application.

Animating the icon **92-12-20**

To animate the icons, use a timer control (which is disabled when the form is not minimized), and change the value of the form's icon property as required.

Calling GetPrivateProfileString() from Visual Basic **93-04-30**

You can call GetPrivateProfileString() (and the other Profile functions) the same way as all other API functions, but with one caveat: you must prefill the string with blanks prior to calling the function.

For example,

```
IniFile = "IniFileName"
Section = "SectioName"
Key = "KeyName"
Default = ""
NumChars = 70
ReturnString = String$(NumChars, " ")

A% = GetPrivateProfileString(Section, Key, Default,
            ReturnString, NumChars, IniFile)
```

Creating controls at runtime in Visual Basic **92-11-15**

To create controls dynamically, you need to create a control array at design time. You can then extend it at runtime, and specify new properties for the new controls you have created.

Disabling automatic variables in Visual Basic 1.0 **93-01-20**

In the tradition of Basic, Visual Basic allows you to use variables which have not been explicitly declared. Unfortunately there is no option to disable this "feature" in Visual Basic 1.0. You can alleviate some of its effects by including a line such as

```
DefStr A-Z
```

in all your modules. This means that Visual Basic will assume all your variables are strings unless explicitly declared, catching at least some typos in your code.

Disabling automatic variables in Visual Basic 2.0 **93-04-25**

In the tradition of Basic, Visual Basic allows you to use variables which have not been explicitly declared. In Visual Basic 2.0 (both Standard and Professional), you can disable this feature by either using the Option Explicit statement at the module level, or everywhere by checking the *Require Variable Declaration* in **Options » Environment.**

Displaying a timed About box **93-04-30**

To display a timed *About* box or copyright notice, use a code fragment similar to the one shown below:

```
AboutBox.show
Delay! = Timer + 2

Do
    Junk% = DoEvents()
Loop Until Delay! > Timer

Unload AboutBox
```

Finding previous instance of a Visual Basic application 93-05-05

The following technique is based on the Visual Basic Professional Edition knowledge base article Q75641; consult that article for more details.

```
Declare Function GetModuleHandle Lib "Kernel" (ByVal lpProgramName As Integer)
    As Integer
Declare Function GetModuleUsage Lib "Kernel" (Byval hProgram As Integer)
    As Integer

Sub MainForm_load
    Dim hw as Integer

    hw = GetModuleHandle("APPNAME.EXE")
    If (GetModuleUsage(hw) > 1) Then
        MsgBox "This program is already running!", 16
        End 'Will kill this instance and leave the running one intact
     End If
End Sub
```

With Visual Basic 2.0, you can also check the App.PrevInstance property in the Form_Load procedure to determine whether a previous instance is already running. If you want to switch to that previous instance, though, you will need to use code similar to the one above.

Passing a structure back to Visual Basic from a DLL

The following description is courtesy of Todd Ogasawara, 1991 (reachable at todd@pinhead.pegasus.com). The code fragments were developed and tested using Borland C++ 2.0 and Microsoft Visual Basic 1.0.

- Define a type that is a pointer to a structure.

```
typedef struct {
    long      fsize;          // file size in bytes
    char      ftime[25];       // last file access time string
} * fileStruct;
```

- Sample function prototype declaration.

```
int FAR pascal FileInfo(char filename[], fileStruct);
```

- Sample DLL C function that receives a filename in a char array from Visual Basic and passes back file size (long) and file modification date (char array) in a structure.

```
// Get file info (access time & size)
int FAR pascal
FileInfo(char filename[], fileStruct far fileinfo)
{
    struct stat statbuf;
    FILE    *stream;
    if (!(stream = fopen(filename, "r"))) {
            return(-1); // ERROR: cannot find named file
    } else {
            fstat(fileno(stream), &statbuf);
            fclose(stream);
    }
    /* file size */
    fileinfo->fsize = statbuf.st_size;
    /* access time */
    strcpy(fileinfo->ftime,ctime(&statbuf.st_ctime));
    return(0);
}
```

- Declare a Visual BASIC "user-defined type" (i.e., a "structure") that matches the structure declared in the DLL C code. See pages 260-261 of the Microsoft Visual BASIC Programmer's Guide for more information about user-defined types.

```
' type (structure) definition in GLOBAL.BAS
Type FileStruct
    Fsize As Long
    Ftime As String * 25
End Type
```

- Declare the function in your GLOBAL.BAS (or whatever you named the file you keep global information in). In this example a function is declared since the DLL C function returns a -1 to indicate an error and a 0 to indicate success. Note that the filename is passed from Visual BASIC to the DLL C function by value (ByVal) while the data in the DLL C structure is passed to Visual BASIC by reference (As). See pages 379-387 of the Microsoft Visual BASIC Programmer's Guide for more information about declaring and calling DLL routines.

```
' declaration in GLOBAL.BAS
Declare Function FileInfo Lib "dosdll.dll" (ByVal FileName$,
    FileInf As FileStruct) As Integer
```

- Example of using the DLL function 'FileInfo' in Visual BASIC.

```
If (Myfile.Filename = "") Then
    Exit Sub
Else
    ThisFileName$ = UCase$(Myfile.Filename)
End If
FileStatus% = FileInfo(ThisFileName$, FileStat)
ThisFileSize$ = Format$(FileStat.Fsize, "###,###,###") + "
    bytes" ThisFileStat$ = Left$(FileStat.Ftime, 24)
```

Right mouse clicks on command buttons **92-12-15**

Visual Basic command buttons do not process right mouse clicks, nor mouse up or down events. To use the right mouse button with a command button, replace the command button with a picture control (use two bitmaps, one for "up" and the other for "down"), which does process these events.

Using Return to move ot the next input field **93-04-30**

If you wish to use the **Return** key to move to the next field (against the Windows user interface guidelines, but expected by many users), you simply need to trap the keys in each control. For example, if you have an array of eight edit controls called *InputField,* you should define a routine like this:

```
InputField_KeyPress(Index as Integer, KeyAscii as Integer)

If (KeyAscii = 13) Then
    If (Index = 7) Then
        InputField(0).SetFocus
    Else
        InputField(Index + 1).SetFocus
    End If
    KeyAscii = 0
End If

End Sub
```

Using Visual Basic strings in a DLL **93-01-20**

To access Visual Basic strings in a DLL, you need to use the functions provided in the Control Development Kit (CDK). You can purchase the CDK separately, but it is also included as part of Visual Basic 2.0 Professional Edition.

As an alternative, passing a Visual Basic string to a DLL as ByVal will automatically convert it to the usual null-terminated ASCII format. However, this will limit the length of the string to 255 bytes.

Visual Basic and Fortran

If you want to use Visual Basic to build a slick interface for your old text-based Fortran code, the approach to take is to build the Fortran code into a DLL, and call it from Visual Basic. You may either pass the parameters as arguments, or you may want to construct a temporary file for more extensive input.

Visual C++

Memory requirements **94-02-22**

Microsoft says that Visual C++ requires 6 MB of memory, and theyre not kidding. Dont attempt to get any work done with 4 MB. Better yet, do yourself a favor and get at least 8 MB if you want to be productive.

If you have relatively limited memory available, you should also specify a large permanent swapfile, probably on the order f 8 MB to 10 MB.

Integrating external makefiles with VC++ **94-02-22**

Visual C++s built-in project files are quite limited, and unable to handle thigs such as clean targets and external (non-VC++) commands. However, switching to an external makefile will not allow you take advantage of VC++s capabilities such as automatic dependency scanning.

The workaround is to create a second makefile, say, "makefile", in the same directory as your project, and then place in the makefile the line

```
!include "project.mak"
```

as well as all the additional targets that you may need. Then add *nmake* with the appropriate file argument to the *Tools* menu, and youre ready to use the full power of makefiles. You can either ask Visual C++ to ask for an argument to this menu item, or you can add multiple menu items for the various targets you may need.

Using version control from Visual Workbench **94-02-22**

While the Visual Workbench is not particularly flexible, its fairly easy to add basic version control capabilities to it. For example you could add the following item to the *Tools* menu:

Command Line:	co
Menu text:	Check Out
Arguments:	$FileName
Initial directory:	$FILEDIR

If you require something more advanced, you can substitute a batchfile or shell script for the command line.

Interfacing to the outside world

Communicating with DOS applications

Clipboard access from DOS applications **93-04-30**

If your DOS applications run with MS-DOS 5.0 or higher, they can access the Windows clipboard (within a Windows DOS session) by using the new function calls in DOS 5. These function calls are described in the Microsoft MS-DOS Programmer's Reference (ISBN 1-55615-546-8, Microsoft Press, $ 27.95).

Determining whether a task is a DOS task **93-01-20**

To determine whether a particular task is a DOS session, you can use the following function, as contributed by Blake Coverett *(blakeco@microsoft.com).*

```
BOOL
IsDosTask(HTASK hTask)
{
    TASKENTRY te = {sizeof(TASKENTRY)};

    if (!IsTask(hTask))
        return FALSE;

    TaskFindHandle(&te, hTask);
    return(!lstrcmp(te.szModule, "WINOLDAP"));
}
```

Passing commandline parameters to DOS applications 93-01-20

The standard WinExec() function will not pass any commandline parameters to the application you are starting. In order to pass parameters, you need to use the ShellExecute() function contained in **shell.dll**, which is a part of the Windows 3.1 SDK.

Passing a pointer to a DOS application or TSR **92-09-15**

In order to pass a pointer to a DOS application (to share memory), you can **not** just pass a Windows pointer. GlobalLock() returns a segment selector table entry, not a physical address. Thus, the simple code below will give you an incorrect address:

```
lpBuffer      = GlobalLock(hBuffer);
InRegs.x.di = FP_OFF(lpBuffer);
SegRegs.es   = FP_SEG(lpBuffer);
int86x(0x7f,&InRegs,&OutRegs,&SegRegs);
```

The problem is that the TSR or DOS application runs in real mode, while Windows applications running in Standard or 386 Enhanced mode use selectors [LDT] and not pointers[SEG:OFF] to access memory.

The following gives an outline of what needs to be done, courtesy of Glenn Boozer (glenn@imagen.com):

To send a pointer to DOS [a Segment:Offset address, not a protected mode selector]

- DosAllocate a memory buffer [This will be a buffer in the first 640K of address space. This buffer is **locked** and will not move. *[Not recomended by Microsoft]*
- Copy the data from the buffer that is in "Windows Application space" into the DOS Buffer
- Get the Segment and Offset of the DOS buffer and pass that to the TSR.
- Release the DOS Buffer

To use a pointer [Segment:offset] you got from a DOS application:

- Allocate a Selector *[Not recomended by Microsoft]*
- Set the selector base and length with the data returned from the TSR. *[Not recomended by Microsoft]*
- Use the data
- Release the selector. *[Not recomended by Microsoft]*

Selected code fragments follow.

```
// Windows kernel calls not in WINDOWS.H
WORD   FAR PASCAL SetSelectorBase(HANDLE hSelector, DWORD dwBase);
WORD   FAR PASCAL SetSelectorLimit(HANDLE hSelector, DWORD dwLimit);
DWORD FAR PASCAL GlobalDosAlloc(DWORD);
WORD   FAR PASCAL GlobalDosFree(WORD);

HANDLE FAR PASCAL
GetPhysicalMemoryHandle()
{
    HANDLE hSel;
    HANDLE hSel2;

    /*. create a selector for use by MakePhysicalMemoryPtr() */
    /* The how of this is taken from an SR response. */
    if ((hSel2 = GlobalAlloc(GMEM_FIXED,(long) 64)) != NULL) {
        hSel = AllocSelector(hSel2);
        GlobalFree(hSel2);
    } else {
        hSel = (HANDLE)NULL;
    }
    return hSel;
```

```
}

LPSTR NEAR PASCAL
MakePhysicalMemoryPtr(WORD wMemHandle, WORD wSegment, WORD wOffset)
{
     /*. set selector base from wSegment parameter */
     SetSelectorBase(wMemHandle, (((LONG)wSegment)<<4) + wOffset );
     /*. set limit for 4K bytes accessable */
     SetSelectorLimit(wMemHandle,(long) 0x0FFF);
     /*. make and return a long pointer using passed wMemHandle */
     return (LPSTR)MAKELONG(0, wMemHandle);
}

void FAR PASCAL
FreePhysicalMemoryPtr(LPSTR lpMemPtr)
{
     FreeSelector(HIWORD(lpMemPtr));
}
```

Main code fragment:

```
// Assuming protected mode
if (!( hPhysMemHandle = GetPhysicalMemoryHandle())) {
     MessageBoxOKHand((LPSTR) "Error: Could not get "
                                  "physical memory for TSR");
     return;
}

// [call tsr-calling routine n times]

// Return the handle we allocated
FreePhysicalMemoryPtr(lpPhysPtr);
```

TSR-calling routine:

```
static FPTR     near fp;

lpsDosParagraphSelector.d =
                GlobalDosAlloc((DWORD)max( APImsg.len, 4));
if(APImsg.buffer) {
     lmemcpy( (LPSTR)MAKELONG(0, lpsDosParagraphSelector.w.sel),
                                  APImsg.buffer, max( APImsg.len, 4));
     fp.w.sel = lpsDosParagraphSelector.w.par;
     fp.w.off = 0;
} else {
     /* Null Pointer */
     fp.p = 0L;
}
dx = fp.w.sel;
bx = fp.w.off;

// Call the TSR
rc = int2f(ax, cx, si, di, dx, bx, (unsigned int far *)&di,
```

```
            (unsigned int far *)&si, (unsigned int far *)&cx,
            (unsigned int far *)&dx, (unsigned int far *)&bx);

(void) GlobalDosFree(lpsDosParagraphSelector.w.sel);
lpPhysPtr = MakePhysicalMemoryPtr(hPhysMemHandle, dx, bx);
fp.p = lpPhysPtr;
```

Starting a Windows application from a DOS session 92-09-15

This is really quite difficult, and you may be happier using an existing implementation (such as *wx* and *wxserver,* which comes with the Windows 3.1 SDK), because in 386Enhanced mode the DOS application and the Windows world are in separate virtual machines; the only context they have in common is the underlying DOS. The basic idea is to use a TSR that
talks to both the DOS app and a Windows "wrapper" app that does the WinExec() for you. Thanks for the explanation are due to Ed Schwalenberg (ed@odi.com).

Create a TSR that gets loaded before Windows is started. Its services will be available to both DOS apps and Windows apps. When Windows is started, your wrapper program can call the TSR with an INT 2F, giving it the address of some GlobalDOSAlloc()'ed memory which will be used to pass information back and forth between the TSR and protected-mode Windows. While processing this INT 2F, you issue one of your own, with AX=1683h, which will return in BX the magic number of the Windows virtual machine (VxD), which is currently 1 but may change.

The DOS application issues an INT 2F, passing the name of the desired application to the TSR. The TSR copies the information into a private data buffer in the TSR's address space, NOT to the GlobalDOSAlloc()'ed memory (which only exists in the Windows virtual machine).

Now for the hard part. You need to call back to Windows when the Windows VM is scheduled. To do that, use INT 2F, AX=1685h, BX=Windows VM number which you saved from the initialization step, CX=flags, DS:SI= priority boost and ES:DI=CS:IP of a procedure to call. When the Windows VM is scheduled, your procedure will be called. That procedure can copy the name of the application into the GlobalDOSAlloc()'ed memory, issue an INT 2F to the windows wrapper program, and IRET. The windows wrapper program can use the data in the GlobalDOSAlloc()'ed memory to WinExec() the desired program.

Multimedia

Checking for a sound card **93-04-30**

To find out whether the system your application is running on has a sound board and the appropriate drivers installed is to call midiOutGetNumDevs() for MIDI drivers and/or waveOutGetNumDevs() for wave drivers. A nonzero result indicates that you can access the respective type of sound card.

MIDI file format **92-10-07**

To find out how to get a copy of the latest MIDI file standard as well as other MIDI-related documentation, send an email message to *mail-server@cs.ruu.nl,* with the following contents:

```
BEGIN
PATH <insert-your-internet-address-here>
HELP
send MIDI/INDEX
END
```

Playing sounds from Visual Basic **92-09-28**

First, you can execute MCI commands by calling the Windows API:

```
Declare Function mciExecute Lib "MMSystem" (ByVal xstr$)
        As Integer

Sub Form_Click ()
    x% = mciExecute("Play c:\windows\ding.wav")
End Sub
```

If all you want to do is play sounds, there is an alternative:

```
Declare Function sndPlaySound Lib "MMSYSTEM" (ByVal snd$, ByVal f%)
        As Integer

Sub Form_Click ()
    x% = sndPlaySound("chime.wav", 0)
End Sub
```

The second argument determines whether the system waits until the sound is finished before returning from the call (0) or returns immediately while the sound is playing (1).

RIFF (DIB, MIDI, RTF and WAV) file formats **93-01-20**

The sound file formats are described in the document **sound10.txt**, available by ftp from *oak.oakland.edu,* directory /pub/misc/sound.

Microsoft's original Resource Interchange File Format (RIFF) descriptions (including WAV, MIDI, DIB and RTF) are available by ftp from *ftp.uu.net,* directory */vendor/microsoft/multimedia,* under filenames **riffmcir.zip** (original spec, RTF format), **riffmcit.zip** (original spec, text format) and **riffnew.zip** (additions to the spec, RTF format). They are also available on Compuserve's WINSDK forum, using truncated 6-character filenames.

They are also available, free of charge, (in paper form) from Microsoft's Developer Services by phoning 800-227-4679, x11771.

Using an accurate timer **92-10-06**

The standard Windows (and PC) timer is only accurate to a disappointing 18ms. If you need a more accurate timer, you should investigate the Windows 3.1 multimedia API, and specifically look at the functions timeBeginPeriod, timeEndPeriod and timeSetEvent.

Miscellaneous

Program Manager DDE interface **93-04-30**

The recommended way of adding applications to Program Manager is to use the DDE interface supported by Windows 3.1 and Windows NT; this is documented both in the SDK reference and in the online help under "Shell Dynamic Data Exchange Overview".

Program Manager group file format **93-05-08**

The Program Manager group file format is documented in the Windows 3.1 SDK reference, volume 4, "Resources", which is included with the SDK, and available separately from Microsoft Press.

You should not read or write these files directly for future compatability; use the Program Manager DDE interface instead.

TWAIN interface specifications **93-04-30**

To get the TWAIN scanner interface specifications, call 800-722-0379 (or call 206-628-5737 and request document 9154 for a faxed order form), or write to:

StarPak, Inc
237 22nd Street
Greeley, CO 80631

The specifications cost approximately $30.

Putting it all together

Compiling and linking

Emulator vs. alternate floating-point math

The alternate math package is faster on non-x87 machines, but slower on those equipped with a math chip. Depending on your application, you might want to ship either, or both. If you need accuracy in floating-point calculations, though, stay away from the alternate math package.

Borland C++ does not support the alternate math package, but it does have a "fast floats" option, which is roughly equivalent.

Emulator floating-point: corrupted code segments

Compiling a Windows application with emulator floating-point causes corrupted code segments when running on a non-8087 equipped system in Windows 2.x and Windows 3.0 Real mode.

The emulated floating point tries to used the coprocessor. When it does not find one on startup, it patches the code to use the software floating point. Patching does not, however, recalculate the code-segment checksum, thus the Windows debugging kernel chokes when it finds that something terrible must have happened to the code. (This problem does not affect Windows 3.x in Standard or 386 Enhanced modes.)

You can get Windows to ignore the checksum errors by setting *EnableSegmentChecksum=0* in the *[debug]* section of **win.ini**; the problem only affects debugging versions of Windows 3.0.

Exporting CDECL functions **92-09-28**

If you specify functions to be exported in the **.def** file, the linker will assume that they use the Windows standard Pascal calling convention. If you didn't declare some of them as PASCAL, you will need to specify those functions in the **.def** file as *_funcname* instead of *funcname,* since C compilers always add an underscore to the front of a name.

Large memory model: why or why not? 93-04-30

Yes, you can do it. There are several problems with using large model, though:

- Your program's data memory will be fixed in real mode. Effectively, your application will cripple any real-mode Windows system. (Of course, this problem doesn't exist with Windows 3.1!)
- Your application will be somewhat larger and run somewhat more slowly, since all your data must be accessed through far pointers. The difference may or may not be significant for your particular application; your best choice is to attempt both medium and large models (if possible), and checking the differences in actual executable size and execution speed.
- You may only be able to have one instance of your application active at any one time. This restriction is imposed by Windows on applications with multiple static data segements: in large model, that means most applications generated with C or C++. Borland C++ 3.0+ and Microsoft C/C++ 7.0 will attempt to keep your static and global data in a single segment (although Microsof C5.1 and C6.0 will not), so as long as that data does not exceed 64K, you could run multiple instances of a large-model application created using those compilers.

You should consider very carefully before you decide that large model is the only way to go; the preferred method is to use medium model, and to allocate far data as required.

Another alternative is to use a compiler such as Watcom C/386 or Zortech C++ for development; this will let you use a single 4GB segment, and 32-bit registers, increasing your applications performace substantially (but limiting it to running in 386 enhanced mode).

Finally, developing for Win32 (which encompasses both the upcoming Windows NT and Win32s) will allow you to use the flat 32-bit memory model without the restrictions and performance penalties associated with using a 32-bit environment on Windows 3.x. Starting with the large model for Windows 3.1 will probably make the transition to true 32-bit code less painful when the time comes to make the move to Windows NT or Win32s.

Using STRICT with windows.h **93-01-20**

The Windows 3.1 **windows.h** file implements optional strict typechecking. You can turn this on by adding the statement

```
#define STRICT
```

before including **windows.h**. Using strict typechecking will cause the compiler to issue warnings if you try to assign a window handle to a variable typed as HDC, for example, as it defines all handles as distinct datatypes. Using strict is probably a good idea to avoid problems when moving to Windows NT and other future versions of Windows later.

Debugging

Debugger stopping at non-existent breakpoints **93-04-30**

If your debugger (CodeView or TDW) is stopping at breakpoints (or Int3s) when you have not set any breakpoints, the likely culprit is your display driver. ATI's Ultra series drivers are known to have this problem, due to their programmers leaving their breakpoints in the "production" drivers.

You can correct the problem by writing down the sequence of bytes surrounding the Int3 (byte CC), exiting Windows, editing the display driver with a binary editor (such as Norton's disk editor), searching for the byte sequence you recorded earlier, and then replacing the CC byte with a 90 (a No-op). *As always, make a backup copy of the driver prior to editing it with a binary editor!*

Debugging a DLL with CodeView **93-04-30**

First, you need to specify the DLL with an /L option on the command line.

If your .exe has debug info, CodeView will find the source. If you do *not* have an .exe with debug information and source code, CVW won't find the sources before you start. In this case, you will need to start CVW, hit F10 a few times to get into the application, and then open the DLL source file, allowing you to set a breakpoint in the DLL.

Probably a better choice is to always include a call to DebugBreak() in the DLL routine you
want to debug, or possibly in the DLL initialization. This will stop at that point, but it will stop *inside Windows,* requiring you to hit F10 a few times to return to your source code.

Dr. Watson log files **93-05-08**

The Dr. Watson log shows the contents of the registers when you application crashed. Even if you can't use it to determine contents of the variables, you can pinpoint the location of the crash.

Make sure you keep a copy of the .map file generated by the linker for the version shipped to your customers; you can then look up the crash location manually from this file when you receive a Dr. Watson log. If you linked a version with /CO /LI, the .map file will also contain line number information, allowing you to pinpoint the line in your program.

You can also use the MAPSYM utility to create a .SYM file. Dr. Watson will consult the .SYM file if it is in the same directory as the .EXE, automatically putting meaningful names in the Dr. Watson log.

Programmer's WorkBench and tab characters **92-09-28**

If you want real tab characters inserted into your source files whenever you press **Tab**, specify *RealTabs:Yes* and *Graphic:Tab* in your **tools.ini** file. You can also quote a single tab character by preceing it with **Ctrl+P**.

Turbo Debugger and Windows 3.1 **92-09-28**

TDW 2.5 and **TDW 3.0** need a new version of **windebug.dll**; otherwise the following error is being displayed while loading a .EXE file in the debugger:

Cannot load WINDEBUG.DLL

The new **windebug.dll** can be found at *ftp.cica.indiana.edu* in the directory /pub/pc/win3/programr/tp as "tpwin31.zip". TDW 2.5 still has a few problems with Windows 3.1 (it sometimes generates exception 13 errors while stepping trough code), but this is a workable solution.

TDW 2.51 (shipped with the Turbo Pascal for Windows 1.0 maintenance release) does not function correctly with Windows 3.1.

TDW 3.1 (shipped with TPW 1.5 and BC++ 3.1) has been written for Windows 3.1 and works correctly under both Windows 3.0 and 3.1.

Turbo Debugger video configuration **92-09-14**

Borland's TDW debugger uses its own video drivers for single-screen Windows debugging. As a result, you must make sure that the debugger can find the correct drivers. As an example, if you are using an ATI Graphics Ultra (or Graphics Vantage), you need to make sure the following lines are in your **tdw.ini** file:

```
[Debugger]
VideoDLL=C:\TPW15\ULTRA.DLL

[VideoOptions]
DebugFile=C:\TDW.LOG
```

Resources and resource tools

Borland C++ Windows tools and Windows 3.1 **92-09-28**

WinSight shipped with BC++ 3.0 does not work under Windows 3.1. There is an update file available at *ftp.cica.indiana.edu* in the directory /pub/pc/win3/programr/bcpp as "wsupd1.zip".

Dr. Frank does not work with Windows 3.1. This program has been reincarnated as an official Borland tool and is shipped with BC++ 3.1 as "WinSpector".

Building a DLL with Zortech C++ **92-11-15**

The key to succesfully building a DLL using Zortech C++ is to make sure that there is a LibMain function; it appears that Borland and Microsoft compilers include it automatically. Simply insert the following fragment into your code:

```
int FAR PASCAL LibMain(HANDLE hInst,     WORD wDataSeg,
                       WORD wHeapSize, LPSTR lpszCmdLine)
{
}
```

Extracting resources from an .EXE file **93-01-06**

There are at least two alternatives: Borland's Resource Workshop (included with most of their Windows tools) allows you to extract and edit individual resources out of executable files. Alternately you can get a program called *rx* (Resource eXtractor) by ftp from *monu6.cc.monash.edu.au* , in directory */pub/win3/programr*.

Help compiler runs out of memory **94-02-22**

If your help files are bigger than the standard help compiler can handle, get the *what* toolkit (available by ftp from *ftp.cica.indiana.edu*), which includes a protected-mode help compiler.

Help development tools **94-02-22**

The following is a partial listing of available help development tools, thanks to Jon Noring. All of these, except for RoboHelp, are available as either hreeware or shareware by ftp; check *ftp.cica.indiana.edu* and other ftp sites.

- Collector
- Create_Help!
- Dr. Help
- Help Development Kit 9.0
- Helper
- HelpMacro
- HyperDoc
- RoboHelp
- RTF-Magic
- WHAT (Windows Help Authoring Tool)

Linking fonts into a .FON file

The linker provided with the Windows 3.0 SDK will produce the following error when linking fonts:

Link Error L2049: no segments defined

The above LINK error is a bug in link. The fix is to run exehdr /r on the .exe file, and then run rc on it. The Windows 3.0 SDK linker incorrectly detects an error, and marks the resulting .exe file with some kind of error bit, even though the rest of the exe file is ok. Exehdr /r will reset this "error bit", after which rc will work just fine.

An alternate fix is to use link4 from Windows 2.x SDK.

Running out of system resources in Visual Basic 92-09-21

Visual Basic is limited to 255 controls per form, but the real limit depends on Windows system resources, which in turn depends on what kind of controls they are and what else you have going on in your system at the time. Here's a simplified explanation:

Windows does indeed give you access to many megabytes of memory (16Mb in Windows 3.0, 256Mb in 3.1). However, some of that memory is "special" because Windows reserves several 64K segments for its own purposes. The most significant of these segments for this discussion are the USER heap and the GDI heap. This is where Windows keeps track of many of its internal data structures -- things like window handles, menus, (USER) and handles to bitmap, clipping regions, and fonts (GDI). (Note that these are handles and related data structures, not the actual menus, bitmaps, or fonts themselves) Collectively,
these two segments are called "Windows resources" and whichever one of the two that is closest to being exhausted is reported as a percentage (of 64K) in the **Help » About** dialog in *Program Manager.*

It should make sense then that as you approach 0% free in one of these two segments "strange things begin to happen" because Windows can't allocate space for new Windows or bitmaps or whatever.

It is unfortunate that there is this 64K limitation: it arises because of Windows' roots in real mode. Future versions (both NT and perhaps a future version of Windows for MS-DOS) will
elminate this. Windows 3.1 was an improvement over 3.0 because it split the single USER heap into several 64K heaps.

Now, getting back to Visual Basic: every form is a window. Every control is a window. Every window consumes some of the USER heap (some more than others). Every bitmap consumes some of the GDI heap; every picture box or form for which you've set *AutoRedraw = True* also consumes additional space in the GDI heap. I'm sure you can see where this is leading: eventually, as you add controls, you run out of system resources. The number at which you run out depends on what the controls are (picture controls chew up resources faster than labels), how many other forms and controls are currently loaded in your application, and how much of your system resources are being consumed by other
applications you may be running at that time.

How to work around this? Only load the forms you need. Unloaded forms (and the controls on them) don't use system resources. Try to reduce the number of picture boxes. Don't set *AutoRedraw* to *True*. If you're trying to display a lot of bitmaps or a grid of data, consider using a custom control designed to do that. The grid control included in the Professional Toolkit allows you to display lots of information without using up the system resources you would if you placed it all in text boxes or labels.

If you are running Windows 3.1 (and you really should be) you can call this function:

```
Declare Function GetFreeSystemResources Lib "User" (ByVal
                    fuSysResource As Integer) As Integer

Global Const GFSR_SYSTEMRESOURCES = &H0
Global Const GFSR_GDIRESOURCES = &H1
Global Const GFSR_USERRESOURCES = &H2
```

This returns the percentage of free system resources For example:

```
Print GetFreeSystemResources(GFSR_SYSTEMRESOURCES)
```

This displays the lower of the GDI or USER heaps (this is exactly what is displayed in those about boxes). If you want to find out how much space is left in USER, use GFSR_USERRESOURCES; if you want to find out how much is left in GDI, use GFSR_GDIRESOURCES. Using this function, you can determine whether you're actually exhausting USER (probably) or GDI (less likely, unless you have a lot of bitmaps).

Thanks for the explanation to Joe Robison (joero@microsoft.com).

Tracking down unfreed resources **92-11-15**

There are several utilities available on *ftp.cica.indiana.edu* which will monitor the heap and memory usage. Look for files ma.zip and ha.zip in */pub/pc/win3/utils.* The Windows 3.1 debug kernel (included with the SDK) also checks for unfreed resources when an application exits. Finally the Heap Walker utility can be used to identify resources in the global heaps.

Screen savers

Creating a Windows 3.1 screen saver **93-04-30**

You can create a Windows 3.1 screen saver by using the **scrnsavelib** library included i nthe Windows 3.1 SDK. The use of the library is described in Chapter 14 of the SDK reference manual, volume 1 (Overviews).

Documentation and help

Adding bitmaps to helpfiles

When you add a bitmap to a help source document using the [bml ...] command, it freqently does not appear in the compiled helpfile. The problem is that the text [bml printer.bmp] is an RTF bitmap inclusion command (which is why you want it there), but Word assumes you really want the literal text "[bml printer.bmp]", and escapes the whole sequence when saving the files as RTF. *Note that the previous description substitutes square brackets for curly brackets to prevent* hc *from actually including those bitmaps. Use curly brackets in your own helpfile source!*

You'll get the actual RTF bitmap inclusion command in the RTF file (and thus a bitmap in the compiled helpfile) by inserting a bitmap using the Word for Windows menu commands and clicking on the "Link to file" checkbox when it asks you which bitmap to insert.

Bullets (and other special characters) in helpfiles **93-04-30**

Using Word for Windows' **Insert » Symbol** for bullets will actually insert sybol references into the helpfile, not actual bullet characters. Your bullets *will* show up in helpfiles if you enter them using the numeric keypad, or paste them in from the Character Map utility. For example, to get the bullet, set your font to Symbol, and type Alt+0183 on the numeric keypad.

The easiest way to determine the actual keycodes is to look up the symbols in the Character Map utility.

Screen Snapshots

To take a snapshot of your screen, just press **PrtScr**, and Windows will copy the image to the clipboard, from where you can paste it into your favourite application. You can also use **Alt+PrtScr** to take a snapshot of only your active pop-up window (child windows such as dialog controls are not counted as "active").

You may wish to select the Monochrome VGA driver prior to doing the screen print to produce 1-bit (2-color) bitmaps for easier printing.

As an alternative, you may wish to use a screen grabber/graphics conversion utility such as HiJaak or PaintShop Pro (see the ***Microsoft Windows FAQ***) if you wish to produce good-quality grayscale bitmaps..

A programmer's bibliography

Windows 3.1 SDK references **93-04-30**

- *Windows 3.1 Programmer's Reference, Volume 1: Overview.* Microsoft Press, 1992, ISBN 1-55615-453-4, $ 29.95
- *Windows 3.1 Programmer's Reference, Volume 2: Functions.* Microsoft Press, 1992, ISBN 1-55615-463-1, $ 39.95
- *Windows 3.1 Programmer's Reference, Volume 3: Messages and structures.* Microsoft Press, 1992, ISBN 1-55615-464-X, $ 29.95
- *Windows 3.1 Programmer's Reference, Volume 4: Resources*. Microsoft Press, 1992, ISBN 1-55615-494-1, $ 22.95
- *Windows 3.1 Guide to Programming.* Microsoft Press, 1992, ISBN 1-55615-452-6, $ 22.95
- *Windows 3.1 Programming Tools.* Microsoft Press, 1992, ISBN 1-55615-454-2, $ 29.95

- *Windows Multimedia Authoring and Tools Guide.* Microsoft Press, 1992, ISBN 1-55615-391-0, $ 24.95
- *Windows Multimedia Programmer's Guide.* Microsoft Press, 1992, ISBN 1-55615-389-9, $ 27.95
- *Windows Multimedia Programmer's Workbook.* Microsoft Press, 1992, ISBN 1-55615-390-2, $ 22.95

- *Windows for Pen Computing Programmer's Reference.* Microsoft Press, 1992

Windows 3.0 SDK references

- *SDK Reference, Volume 1: Functions and messages.* Microsoft Press, 1990, part no. 06856
- *SDK Reference, Volume 2: Resource scripts and file formats.* Microsoft Press, 1990, part no. 06857
- *SDK Guide to Programming.* Microsoft Press. 1990, part no. 06854
- *SDK Tools.* Microsoft Press, 1990, part no. 06854
- *SAA CUA Advanced Interface Design Guide.* IBM, 1989, part no. SC26-4582-0

Win32 (Windows NT) API references **93-04-30**

- Microsoft: *Win32 Programmer's Guide v.1: Systems Services and Windows Management.* Microsoft Press, 1993, ISBN 1-55615-515-8, $39.95
- Microsoft: *Win32 Programmer's Guide v.2: Graphics Device Interface.* Microsoft Press, 1993, ISBN 1-55615-516-6, $39.95
- Microsoft: *Win32 Programmer's Reference v.1: API Functions (A-M).* Microsoft Press, 1993, ISBN 1-55615-517-4, $39.95
- Microsoft: *Win32 Programmer's Reference v.2: API Functions (N-Z).* Microsoft Press, 1993, ISBN 1-55615-518-2, $39.95
- Microsoft: *Win32 Programmer's Reference v.3: Messages, Structures and Data Types.* Microsoft Press, 1993, ISBN 1-55615-519-0, $39.95

Windows user interface guidelines

- *The GUI Guide.* Microsoft Press, 1993, ISBN 1-55615-538-7, $ 29.95
- *The Windows Interface: An Application Design Guide.* Microsoft Press, 1993, ISBN 1-55615-439-9, $ 39.95.
- *Windows 3.1 User Interface Guidelines.* [for Windows 3.1, Pen Windows and Windows NT] Microsoft Press, 1992
- *SAA CUA Advanced Interface Design Guide.* [for Windows 3.0 and OS/2 1.x] IBM, 1989, part no. SC26-4582-0
- *SAA CUA'91 Design Guide.* [for OS/2 2.0] IBM, 1991, part no. SC34-4289, $10.00
- *SAA CUA'91 Reference.* [for OS/2 2.0] IBM, 1991, part no. SC34-4290, $18.25

Microsoft Technical Notes **93-08-13**

Microsoft's Technical Notes are available on the Microsoft Developer Network CD-ROMs, by ftp at *ftp.uu.net,* directory */vendor/microsoft/developer-network,* or from the WINSDK forum on CompuServe. The following is a quick index to the numbered files available at *uunet.*

File	Name	Description
3-1	Back	Building a DLL using large model
3-2	CallB	C++ class member functions as callbacks
3-3	ClsExp	C++ class export syntax
3-4	NewOpr	C++ new operator in large model
3-5	Owner	C++ new operator in DLLs
3-6	Smart	Using _fmalloc with Windows
3-7	Zusammen	Using C/C++ compiler options
3-8	AppExec	Launching and waiting for Windows applications
3-9	DDEExec	Creating a DDE Execute server
3-10	DDERecon	Creating a DDE hot link client
3-11	DDEServer	Creating a DDE server
3-12	PmgrAPI	DDE command string interface to Program Manager
3-13	StockSrv	DDE hot links
3-14	Glyph	TrueType font engine and GetGlyphOutline()
3-15	Lava	Palette animation and pop-up menus
3-16	MergeDIB	Merging DIBs
3-17	MultiPal	Multiple palette management
3-18	TransBlt	Transparent bitmaps
3-19	Tri	Drawing to screen or memory DIB
3-20	TriQ	Drawing to screen or memory DIB
3-21	CountDOS	Combining a Windows application and a DOS TSR
3-22	WinFloat	Advanced floating-point functionality in Windows
3-23	Detect	Detecting presence of multimedia extensions
3-24	JoyToy	Using joystick services
3-25	MMPF	Reads and analyzes .MMP files
3-26	MMPlay	Plays .MMP files
3-27	MMSys	Using multimedia without drivers
3-28	WaveConv	Converting .WAV and .RIF files
3-29	Patron	OLE client
3-30	Schmoo	OLE server
3-31	Ctl3D	Implementing 3D dialogs and controls, a la Excel 4.0
3-32	Dialogs	Using common dialogs
3-33	EdAlign	Changing edit control alignment
3-34	MinMax	Controlling window minimum and maximum size
3-35	StatBar	Creating a status bar
3-36	Styles	Using user-selected styles for a window
3-37	VList	Using virtual listboxes

File	Name	Description
4-1	Blocks	Dragging and stretching with MFC
4-2	MFCDIB	DIBs with MFC
4-3	Modeless	Modeless dialogs with MFC
4-4	Subclass	Subclassing edit controls with MFC
4-5	BallCli	DDE client using DDEML
4-6	BallSrv	DDE server using DDEML
4-7	DMLClt	Basic DDEML client
4-8	DMLSrv	Basic DDEML server

4-9	BMUtil	Bitmap techniques and utilities for MFC
4-10	DragBMP	Dragging bitmaps smoothly with MFC
4-11	FADE	Fading bitmaps using palette animation
4-12	GDIRsrcs	Extracting resources from resource files
4-13	GDIWatch	Using TOOLHELP.DLL to walk heap
4-14	RLEApp	Animating with DIB RLE format
4-16	WinCap	Capturing screens using DIB API
4-17	AsmClock	Windows programming with MASM 6.0
4-18	Backgrnd	Background processing
4-19	MakeApp	Generating a Windows application
4-20	ShowGrp	Reading group files
4-21	VerStamp	Uses VER.DLL to get version information for a file
4-22	WinQuery	Querying the SQL Server
4-23	Xtension	Extension DLL for File Manager
4-24	BadApp	"Bad Application" handling
4-25	DLLFloat	Converting floating point to character strings
4-26	DLLSkel	Medium-model DLLs
4-27	Fault	Exception and Fault Trapping
4-28	Handle	Validating global and local handles
4-29	MultApp	Multiple instance DLL with separate data blocks
4-30	TermWait	Launching a task and waiting for completion
4-31	MIDIKeyb	Using the midiKeyB control
4-32	ZYZGauge	Implementing a progress bar
4-33	ALCKey	Uses ALC values to filter input
4-34	AnnoPrnt	Annotating and printing text and ink
4-35	Annotate	Adding handwriting annotation to a text file
4-36	DynBedit	Changing an edit control to bedit dynamically
4-37	Grid	Using pen gestures
4-38	Hotspots	Reconizing gesture hotspots
4-39	Parser	Mapping raw data into characters
4-40	Pressure	Capturing pen pressure information
4-41	RCDump	Accessing the RCRESULT Structure
4-42	View	Displaying ink slowly
4-43	Health	Visual Basic Pen Windows healthcare application
4-44	Insure2	Visual Basic Pen Windows Insurance application
4-45	LoanApp	Visual Basic Pen Windows loan application
4-46	Receipt	Visual Basic Pen Windows order application
4-47	SaleAnal	Visual Basic Pen Windows forecast tracking
4-48	CollColr	Changing System Colors
4-49	DynDlg	Implementing a dynamic options dialog
4-50	EXEView	Extracting and decoding information from an .EXE
4-51	HitTest	Testing for click on an icon
4-52	HotKey	Installing hotkeys
4-53	MDIDlg	Using dialogs as MDI windows
4-54	SignOn	Creating a sign-on screen
4-55	SysParam	Using the SystemParametersInfo() function

File	**Name**	**Description**
10-1	AppExec	Launching other Windows-based applications
10-2	BigBit	Using TrueColor devices
10-3	Client	OLE Client Implementation Guide 1.02
10-4	Ctl3D	Adding 3-D effects to controls
10-5	CtlDlgEd	Developing custom controls for the dialog editor
10-6	DDEDLL	Performing DDE from a DLL
10-7	DDEHotLk	Using DDE hot links

10-8	DDEOLE	Supporting clipboard, DDE, and OLE
10-9	DDESysTp	Supporting the DDE system topic
10-10	DevEnv	Establishing a flexible development environment
10-11	DIBPal	Using DIBs with Palettes
10-12	DIBs2	Using DIBs
10-13	DLLIntro	Introduction to DLLs
10-14	EditCtls	Edit control reference
10-15	Faults	Bulletproofing functions with TOOLHELP.DLL
10-16	FontMap	Windows font mapping
10-17	GDIOver2	Windows GDI overview
10-18	GetGlyph	Advanced TrueType: GetGlyphOutline() documentation
10-19	Hooks	Using Windows' hook functions
10-20	JSConcpt	OLE: A short overview
10-21	LargeM4	Using large model
10-22	Listbox	Listbox control reference
10-23	Listhrz2	Using horizontal scroll bars in listboxes
10-24	Mapping	Coordinate mapping in Windows
10-25	Metafile	An overview of metafiles
10-26	MinMax	Usign WM_GETMINMAXINFO
10-27	MTI	An overview of modules, tasks and instances
10-28	Objects	Using GDI objects
10-29	PalAware	Creating a palette awareness
10-30	Palette	An introduction to the palette manager
10-31	Porting	Porting 16-Bit Windows Applications to Win32
10-32	Primitiv	Device-independent graphics primitives
10-33	Print	An introduction to printing
10-34	RawDDE	DDE transaction definition tables
10-35	Server	OLE server implentation guide
10-36	StatBar	Implementing a Status Bar
10-37	StatiCtl	Static control reference
10-38	Streams	OLE Streams introduction
10-39	Styles	Window hierarchies and styles
10-40	Subclass	Safe subclassing
10-41	T2API	Using TrueType
10-42	Timer2	Timers and timing in Windows
10-43	TransBlt	Transparent BitBlits
10-44	TrueType	An introduction to TrueType
10-45	TSR-Supp	TSR support in Windows 3.1
10-46	TT	An introduction to TrueType
10-47	TTFonts	Linear and nonlinear scaling
10-48	Use-Cust	Using and customizing common dialogs
10-49	VLB	Virtual listboxes
10-50	VxDLite	A miniature DDK for creating VxDs
10-51	WEP	Loading, initializing, and terminating a DLL
10-52	Win32DLL	DLLs in Win32
10-53	WinFloat	Floating point in Windows
10-54	PenUI	Pen Windows user interface guidelines
10-55	SymbolGr	Using the symbol graph
10-56	W4PRecog	Moderating the Pen Windows recognition process
10-57	GraphX	Graphics design and optimization
10-58	Memory	Optimizing memory usage and performance
10-59	MMAware	Creating multimedia-aware applications
10-60	MsftMM	Microsoft multimedia document overview
10-61	OptCDRom	CD-ROM design and optimization
10-62	RiffNew	New multimedia types and techniques

10-63	Tech1	Multimedia technical support update
10-64	Video	Multimedia video techniques
10-65	Dr_GUI1	Ask Dr. GUI # 1
10-66	Dr_GUI2	Ask Dr. GUI # 2
10-67	Dr_GUI3	Ask Dr. GUI # 3
10-68	Dr_GUI4	Ask Dr. GUI # 4
10-69	Dr_GUI5	Ask Dr. GUI # 5
10-70	Dr_GUI6	Ask Dr. GUI # 6

File	**Name**	**Description**
11-1	CallB	C++ class member functions as callbacks
11-2	Cbl_CApp	Passing parameters between C and COBOL modules
11-3	CPPDLL	Exporting an entire C++ class
11-4	Draft3	Using MOVE in MS-DOS
11-5	Fangle	Using *new* in C++
11-6	Malloc	Using _fmalloc with Windows
11-7	ObjMapC	Object mapping in C++
11-8	Optim	Setting C/C++ compiler options
11-9	SS208	Relocatable Object Module format specification
11-10	TN0001	MFC: WNDCLASS structures
11-11	TN0002	MFC: Presistent C++ storage
11-12	TN0003	MFC: Mapping Windows handles to C++ objects
11-13	TN0004	MFC: Using template classes in C++
11-14	TN0005	MFC: Using MDI
11-15	TN0006	MFC: Using message maps
11-16	TN0007	MFC: Windows debugging and trace options
11-17	TN0008	MFC: general OLE overview
11-18	TN0009	MFC: creating OLE clients
11-19	TN0010	MFC: creating OLE servers
11-20	TN0011	MFC: creating DLLs
11-21	TN0012	MFC: using Windows 3.1 robustness features
11-22	TN0013	MFC: using standard dialog classes
11-23	TN0014	MFC: creating custom controls
11-24	TN0015	MFC: Pen Windows
11-25	TN0016	MFC: multiple inheritance
11-26	Tricks	New tricks for the C/C++ compiler
11-27	BuildSwi	Programmer's Workbench build switches
11-28	DLL	Creating DLLs with QuickC for Windows

File	**Name**	**Description**
13-1	DynaComm	Developing DDE applications with DynaComm
13-2	New20	New WordBasic features in Word for Windows 2.0
13-3	Prat1	Data mangement with Word for Windows and PackRat
13-4	QPlusE	Data exchange with Q+E
13-5	RealTips	EchoOff for WordBasic macros
13-6	TJTips	Private initialization files with Word for Windows
13-6	XTalkDDE	DDE with CrossTalk for Windows

Programming guides: general **93-07-30**

Special notice must go to Charles Petzold's Windows bible, which is shown first. All other Windows programming guides are listed alphabetically by author.

- Petzold, Charles: *Programming Windows*, 3rd edition *(with disk).* Microsoft Press, 1990, ISBN 1-55615-395-3, $49.95 (with 3.5" disk)

- Baer, Jürgen: *Introduction to WIndows 3.1 Programming.* Abacus, 1992, $ 34.95
- Baer, Jürgen: *Windows 3.1 Intern (with disk).* Abacus, 1992, $ 49.95
- Clark, Jeffrey: *Windows Programmer's Guide to OLE/DDE (with disk).* SAMS, 1992, $ 34.95
- Custer, Helen: *Inside Windows NT.* Microsoft Press, 1992, ISBN: 1-55615-481-X
- Farrell, Tim: *Programming in WIndows 3.1,* 2nd edition *(with disk).* Addison-Wesley, 1992, $29.95
- Kauler, Barry: *Windows Assembly Language and Systems Programming.* Prentice Hall, 1993.
- Klein, Mike: *Windows Programmer's Guide to DLLs and Memory Management (with disk).* SAMS, 1992, $ 34.95
- Heller, Martin: *Advanced WIndows Programming.* John Wiley & sons, 1992, ISBN 0-471-54711-5, $ 32.95
- Leavens, Alex: *Windows Programmer's Guide to Resources (with disk).* SAMS, 1992, $ 34.95
- McCord, James: *Windows 3.1 Programmer's Reference.* Que, 1992, $ 39.95
- Microsoft: *The GUI Guide.* Microsoft Press, 1993, ISBN 1-55615-538-7, $ 29.95
- Monk. Tim: *Windows Programmer's Guide to Serial Communications.,* SAMS, 1992, ISBN 0-672-30030-3, $ 39.95
- Myers, Brian and Chris Doner: *Programmer's Introduction to Windows 3.1 (with disk).* Sybex, 1992, ISBN 0-7821-1034-7, $ 34.95
- Norton, Peter and Paul Yao: *Windows 3.1 Power Programming Techniques.* Bantam Books, 1992, $29.95
- Rector, Brent: *Developing Windows 3 Applications Microsoft Windows SDK.* SAMS, 1992, $ 29.95
- Richter, Jeffrey M.: *Windows 3: A Developer's Guide (with disk).* M&T Books, 1991, ISBN 1-55851-164-4
- Southerton, Alan: *Windows 3.0 Programming Primer (with disk).* Addison-Wesley, 1990, $ 34.95
- Wilken, Peter: *Windows System Programming (with disk).* Abacus, 1991, $ 39.95
- Wilton, Richard: *Microsoft Windows 3 Developer's Workshop.* Microsoft Press, 1991, $24.95
- *Windows 3.1 Developer's Workshop).* Microsoft Press, 1992, ISBN 1-55615-480-1, $ 34.95

Programming guides: class libraries **93-01-22**

- Dilascia, Paul: *Windows++.* Addison-Wesley, 1992, $29.95
- McCord, James: *Developing Windows Applications With Borland C++ 3.0,* SAMS, 1992, ISBN 0-672-30231-4, $ 39.95.
- Norton, Peter and Paul Yao: *Borland C++ Programming for Windows.* Bantam Books, 1992, $29.95
- Roetzheim, William: *Programming Windows with Borland C++.*
- Shammas, Namir: *Windows Programmer's Guide to Object Windows Library (with disk).* SAMS, 1992, $ 34.95
- Shammas, Namir: *Windows Programmer's Guide to Microsoft Foundation Classes (with disk).* SAMS, 1992, $ 34.95
- Heiny, Loren: *Windows Graphics Programming with Borland C++.* Wiley, 1992, $ 29.95

Programming guides: device drivers and internals 93-07-30

- Custer, Helen: *Inside Windows NT.* Microsoft Press, 1993, ISBN 1-55615-481-X, $24.95
- Norton, Daniel A.: *Writing Windows Device Drivers.* Addison-Wesley, 1991, $29.95
- Kauler, Barry: *Windows Assembly Language and Systems Programming.* Prentice Hall, 1993.
- Schulman, Andrew: *Undocumented Windows (with disk).* Addison-Wesley, 1992, $ 39.95

Programming guides: Visual Basic **93-04-30**

- Nelson, Ross: *Running Visual Basic for Windows.* Microsoft Press, 1993, ISBN 1-55615-477-1, $ 22.95
- *Visual Basic How-To.* Waite Group Press, 1992
- Young, Michael J.: *Visual Basic Game Programming for Windows.* Microsoft Press, 1992, ISBN 1-55615-503-4, $ 39.95

Programming guides: macro languages **93-01-22**

- Borland, Russell: *Word for Windows Macros.* Microsoft Press, 1992, ISBN 1-55615-486-0, $ 34.95
- Kyd, Charles W., and Chris Kinata: *Microsoft Excel Macros.* Microsoft Press, 1992, ISBN 1-55615-526-3, $ 29.95
- Leonhard, Woody: *Windows 3.1 Programming for Mere Mortals.* Addison-Wesley, 1992, $ 34.95
- *The Power of Word 2 for Windows macros.* MIS Press, 1992

Magazines **92-09-21**

- *BasicPro Magazine*
  299 California Ave. S120, Palo Alto, CA 94306-1912
  (415) 688-1808
  Not Windows-specific, but with extensive Visual Basic coverage
- *Dr. Dobbs' Journal*
  411 Borel Ave., San Mateo, CA 94402-3522
  (800) 456-1215, (303) 447-9330
  Technical. Moderate Windows coverage.
- *Inside Visual Basic*
  9420 Bunsen Parkway, suite 300, Louisville, KY 40220
  (800) 223-8720, (502) 491-1900
- *Microsoft Systems Journal*
  501 Galveston Dr., Redwood City, CA 94063
  (415) 366-3600
  Technical. Extensive Windows coverage.
- *Windows Magazine*
  600 Community Community Dr., Manhasset, NY 11030
  (800) 248-3584, (303) 447-9330
  Non-technical.
- *Windows/DOS Developer's Journal*
  2601 Iowa Rd., Lawrence, KS 66046
  (913) 841-1631
  Technical. Good Windows coverage.

Microsoft Developers' Network **93-04-30**

The Microsoft Developers' Network subscription includes a bimonthly newsletter with some useful information, as well as a CD-ROM subscription, packed with source code samples, technical notes, all the free SDKs (such as MAPI and LSAPI), Microsoft Systems Journal sources, Microsoft Knowledge Bases for their development tools and a complete version of Charles Petzold's *Programming Windows* online.

While the subscriptions are not cheap at $199/year, no professional Windows programmer (read: anyone making a living programming for Windows) should be without one. You will quickly make back the cost with higher productivity.

Magazine source code availability 93-04-30

- **Dr. Dobbs' Journal**
  simtel20.army.mil, wuarchive.wustl.edu
  Also available on CompuServe
- **Microsoft Systems Journal**
  simtel20.army.mil, wuarchive.wustl.edu (old issues only)
  Also available on Microsoft OnLine and CompuServe
  Also available on the Microsoft Developers' Network CD-ROM
- **Windows/DOS Developer's Journal**
  ftp.uu.net: /published/windowsdos/19YY/monYY.zip