Getting Started

Microsoft® WINDOWS™ SOFTWARE DEVELOPMENT KIT

Microsoft® Windows™ Software Development Kit

Version 3.1

Getting Started

For the Microsoft Windows Operating System

Microsoft Corporation

Document No. PC28914-0492

Contents

Introduction

The Microsoft® Windows™ 3.1 Software Development Kit (SDK) is a set of libraries, header files, tools, books, online help, and sample source programs designed to help you create Windows applications.

This guide introduces version 3.1 of the SDK and covers the following topics:

- Installing the SDK software
- Preparing your computer for Windows-application development
- Reviewing what is new for Windows version 3.1
- Determining how changes may affect your existing Windows applications
- Building Windows applications for both Windows 3.0 and Windows 3.1
- Creating robust Windows applications by using strict type-checking and parameter validation

Software Development Kit Contents

The SDK contains the tools and information for using the following in applications:

- Windows application programming interface (API)
- Windows extensions, such as common dialog boxes and object linking and embedding (OLE)
- Windows Help
- Microsoft Windows Setup Toolkit (Windows Setup)
- Microsoft® Windows™ operating system with Multimedia Extensions
- Microsoft® Windows™ for Pen Computing

The SDK is provided on both CD-ROM and 3.5-inch, 1.44-megabyte disks. (5.25-inch, 1.2-megabyte disks are available via fulfillment.) You install the SDK by using the SDK installation program, Install, described in Chapter 1, "Installing the Software Development Kit."

Redistributable Libraries and Files

Many new Windows features are included in dynamic-link libraries (DLLs) and can be used in Windows 3.0 and Windows 3.1. If you use the new features in your Windows 3.0 applications, you will need to redistribute the Windows DLLs and related files that support these features. The SDK includes the following redistributable files:

File	Description
COMMDLG.DLL	Common dialog box library. The SDK also includes 10 other DLLs with the filename COMMDLG but with different extensions. These are international versions of the library.
DDEML.DLL	Dynamic Data Exchange Management Library
DIB.DRV	Device-independent bitmap driver
EXPAND.EXE	File Expansion Utility
LZEXPAND.DLL	Data decompression library
MCIPIONR.DRV	Multimedia control interface (MCI) driver for Pioneer Videodisc
OEMSETUP.INF	Installation file for MCIPIONR.DRV
OLECLI.DLL	Object linking and embedding client library
OLESVR.DLL	Object linking and embedding client library
PENWIN.DLL	Microsoft Windows for Pen Computing library
REGEDIT.EXE	Microsoft Windows Registration Editor
SHELL.DLL	Registration database and drag-drop support library
SMALL*X*.FON	Small bitmap fonts for use in print-preview applications. SMALLB.FON is for EGA resolution; SMALLE.FON is for VGA resolution; SMALLF.FON is for IBM 8514/a resolution.
TOOLHELP.DLL	Tool helper library
VER.DLL	File installation and version checking library
VTD.386	Virtual timer device (supports tool helper library)
WINHELP.EXE	Microsoft Windows Help version 3.1
WINHELP.HLP	Help for Windows Help version 3.1
WINMEM32.DLL	Windows 32-bit memory-management library

Note Many files associated with Windows-based setup programs (Windows Setup) are also redistributable but are not listed here. For more information about the redistributable libraries and files, please see the separate SDK license agreement.

Books in the Software Development Kit

The SDK provides thousands of pages of information about Windows and Windows programming. This quantity of information can be intimidating to both the novice and the experienced programmer, but if you take a few moments to review the following descriptions of the SDK books, you'll be able to determine what path to take through the information.

Microsoft Windows Guide to Programming provides a step-by-step explanation of how to create simple Windows applications that use the basic features of Microsoft Windows. If you are new to programming for Windows, read this book first.

Microsoft Windows User Interface Guidelines describes the recommended user interface for Windows applications. Following these guidelines will help you enjoy the benefits of having your application's interface be consistent with other Windows applications.

Microsoft Windows Programmer's Reference, Volume 1: Overview explains and illustrates the basic concepts of the Windows API and the Windows extensions. If you want an in-depth look at programming for Windows, read this book.

Microsoft Windows Programmer's Reference, Volume 2: Functions provides details for the functions of the Windows API and the Windows extensions.

Microsoft Windows Programmer's Reference, Volume 3: Messages, Structures, and Macros provides details for the types, structures, messages, macros, and printer escapes used by the Windows API and the Windows extensions.

Microsoft Windows Programmer's Reference, Volume 4: Resources provides complete details for the format of files and resources used by Windows applications.

Microsoft Windows Programming Tools explains how to use the various SDK tools to create resources for a Windows application.

Microsoft Windows Multimedia Programmer's Guide explains how to use the Windows Multimedia API in your Windows applications.

Microsoft Windows Multimedia Programmer's Reference provides details for the functions of the Windows Multimedia API.

Microsoft Windows for Pen Computing Programmer's Reference provides details for the functions of the Microsoft Windows for Pen Computing API.

Microsoft Windows Setup Toolkit explains how to create a Windows-based setup program for your Windows application.

The Win32™ Application Programming Interface: An Overview describes the Win32 application programming interface and explains how to create applications for Windows 3.1 that can be easily ported to Win32.

Online Information

Much of the information in the SDK books is also available online in either Microsoft QuickHelp (QH.EXE) or Microsoft Windows Help (WINHELP.EXE) formats. The SDK includes the following online information files:

File	Description
MCISTRQH.HLP	Multimedia Control Interface Reference (QuickHelp)
MCISTRWH.HLP	Multimedia Control Interface Reference (Windows Help)
PENAPI_A.HLP	Microsoft Windows for Pen Computing Programmer's Reference (QuickHelp)
PENAPI_W.HLP	Microsoft Windows for Pen Computing Programmer's Reference (Windows Help)
WIN31MQH.HLP	Microsoft Windows Multimedia Reference (QuickHelp)
WIN31MWH.HLP	Microsoft Windows Multimedia Reference (Windows Help)
WIN31QH.HLP	Microsoft Windows Programmer's Reference (QuickHelp)
WIN31WH.HLP	Microsoft Windows Programmer's Reference (QuickHelp)

Debugging with the Software Development Kit

The SDK includes many new functions and tools that you can use in conjunction with the debugging version of Windows to produce trouble-free code. Every Windows application should be thoroughly tested on the debugging version of Windows and should take advantage of the STRICT type-checking that is available with the WINDOWS.H header file. Applications should compile without warnings before they are released.

For more information about the many enhancements that will help you develop robust applications for Windows version 3.1, see Chapter 3, "Creating Windows Applications."

The Microsoft Connection

The Microsoft Connection on CompuServe provides online technical information for Microsoft products, including the SDK. With The Microsoft Connection, you can exchange messages with Microsoft professionals and experienced Microsoft users, and you can download free software, such as drivers, patches, tools, and add-ons, provided by Microsoft and CompuServe members.

By using The Microsoft Connection, you can access the Microsoft Developer Services area. You are encouraged to use this area to speak directly to Microsoft about developer-related issues. The Microsoft Developer Services area offers the following advantages:

- Developer Forums. These forums cover information about Windows, languages, tools, and utilities from a developer's perspective. For example, the Windows SDK Forum provides information about programming for Windows. The section leads for these forums are from Microsoft Product Support and can help answer your questions about the Windows API.
- Confidential Technical Service Requests. Microsoft offers private (fee-based per incident) technical support to help solve your more complex development problems. For more details, see the Microsoft Developer Services area.
- Developer Knowledge Base. This up-to-date reference tool, compiled by Microsoft Product Support, contains developer-specific technical information about Microsoft products.
- Software Library. This collection of text and graphics files, sample code, and utilities can be searched by keyword, and the files can be downloaded for local use.

To connect to The Microsoft Connection, type **GO MICROSOFT** at the CompuServe "!" prompt. For information about establishing a CompuServe account, call (800) 848–8199, 8:00 A.M. to 10:00 P.M. EST. Ask for operator 230 and receive a $15 connect-time usage credit.

Installing the Software Development Kit

Chapter 1

This chapter explains how to install the Microsoft Windows version 3.1 Software Development Kit (SDK) and how to set up your computer for application development.

1.1 Hardware and Software Requirements

You can install and use the SDK on any computer that meets the minimum qualifications for Windows version 3.1. However, for application development, the following hardware is recommended:

- A personal computer using an 80386 or higher microprocessor
- 4 megabytes of conventional and extended memory
- One 3.5-inch (1.44 megabyte) disk drive or a CD-ROM drive.
- One hard disk with at least 40 megabytes available space
- A graphics adapter and compatible monitor
- A Microsoft® Mouse or compatible pointing device (required for some development tools)

In addition to a computer for application development, many developers use one or more extra computers to test their applications under different memory and display configurations. Although extra computers are not required, they are recommended.

Before you install the SDK, be sure that the following software is already installed on your system:

- Microsoft® MS-DOS® operating system version 3.1 or later
- Windows version 3.1
- Any compiler or assembler (and accompanying linker) that supports Windows application development

1.2 Using the Install Program

You install the SDK by using Install, the SDK installation program. This program decompresses and copies the SDK software from the SDK disks to your hard disk; all files on the SDK disks are in compressed format. Install also adds the group Software Development Kit 3.1 to your Program Manager window and adds information to the initialization files in your Windows directory. Optionally, Install copies the sample program sources, online information files, redistributable libraries, Windows for Pen Computing files, and the files for Windows Setup.

Install is an MS-DOS program that you must run from the MS-DOS command line. You cannot install the SDK while Windows is running, so make sure you exit Windows before running Install.

To install the SDK, follow these steps:

1. Insert Disk 1 or the CD-ROM disc into the appropriate drive.
2. At the MS-DOS prompt, change to the drive containing Disk 1. For example, type **a:** and press ENTER to change to drive A.
3. For 3.5-inch disks, type **install** and press ENTER. For CD-ROM, type **cd \install**, then type **install** and press ENTER.
4. Follow the online instructions to complete the installation. You should install the SDK in a new directory to avoid confusing new files with files from other development environments.

Install will ask whether you want the following optional components installed:

- Redistributable libraries and files
- Windows online information, Microsoft QuickHelp (QH.EXE) format

- Windows online information, Microsoft Windows Help (WINHELP.EXE) format
- Sample sources
- Windows for Pen Computing files
- Windows Setup files

To help you decide, Install displays information about the component and prompts for a Yes or No answer. If you direct Install not to install an optional component, you can always install that component later by reinstalling the SDK.

If you have Microsoft® C Optimizing Compiler version 6.0, you must install special C run-time libraries and header files from a separate C installation disk that is included in the SDK. The C libraries contain run-time functions specially modified for use with Windows. The C run-time header files are Windows-compatible versions of the standard C header files. These Windows header files are identical to the standard C header files but contain **ifdef** directives that hide function prototypes for functions that are not compatible with Windows.

If you have Microsoft® QuickC® for Windows™, version 7.0 of the Microsoft C Optimizing Compiler, or another compiler that is compatible with Windows, you have these C libraries and header files already and need not install new ones.

You use the INSTALL.BAT batch file to install the libraries and header files. Insert the C run-time libraries disk in drive A or B and type a command of the following form:

install [*drive*:*path**lib-directory*] [*drive*:*path**include-directory*]

1.3 Updating Your System Files

Once the installation is complete, you must update your AUTOEXEC.BAT file and may need to update your CONFIG.SYS file.

In your AUTOEXEC.BAT file, update the PATH, LIB, and INCLUDE environment variables so that they include the directories containing the Windows tools, libraries, and header files. For the names of the directories to include, see the README.TXT file. Install customizes README.TXT according to the directories you specified during installation and puts it in your directory for Windows development tools.

When you installed Windows version 3.1, your CONFIG.SYS file should have been updated to contain the correct settings for files and buffers. If not, add the following statements to your CONFIG.SYS file:

```
files=30
buffers=20
```

If you are using the SMARTDrive disk-caching utility, you may want to set **buffers** to **5**. With SMARTDrive in use, setting **buffers** to **20** uses memory without increasing your system's speed.

After updating the CONFIG.SYS or AUTOEXEC.BAT file, you must restart your computer for the changes to take effect.

1.4 Setting Up Your Windows Development Environment

You can use Windows as your primary application-development environment. Windows allows you to switch quickly among applications, including non-Windows applications. As a result, you can write the code for your application, create its resources (such as dialog boxes), compile and link the application, and debug and test it, all in the same Windows session.

To provide the best possible performance for creating Windows applications, you should follow these tips:

- Configure Windows to improve the performance of your software-development tools.
- Associate file extensions with software-development tools when you are using File Manager.
- Compile and link in 386 enhanced mode.
- Compact your hard disk to improve disk-access times.

When you associate a filename extension with a development tool, you can run the tool simply by choosing the appropriate filename in the File Manager directory window. File Manager starts the application and passes the chosen filename to the application.

To associate a filename extension with an application, choose the Associate command from the File menu in File Manager. The system records the filename extension and associated application in the WIN.INI file and in the registration database.

1.5 Using Online Information

After the online information files have been installed, you can view the SDK information in Windows Help files by using Windows Program Manager. Open the Software Development Kit 3.1 group and choose the help icon of the information you want to view. Alternatively, you can start Windows Help and use the Open command in the File menu to specify the help file you want to view.

To view SDK information in QuickHelp files, set the HELPFILES environment variable to the name of the directory containing the SDK help files, and then start QuickHelp. QuickHelp loads all help files, so to view the information from a specific help file, select the information from the list in the Categories menu.

What's New for Windows Version 3.1?

Chapter 2

This chapter describes many of the enhancements and changes to the Microsoft Windows 3.1 Software Development Kit (SDK). The chapter summarizes the major differences between this SDK and the SDK for Windows version 3.0 and provides information about how some changes may affect your existing applications.

Several major areas of new growth for Windows 3.1 are not discussed in this chapter. Specifically, the chapter does not discuss the Microsoft Windows operating system with Multimedia Extensions, Microsoft Windows for Pen Computing, or the creation of windows-based setup programs. For information about these topics, see the *Microsoft Windows Multimedia Programmer's Guide*, the *Microsoft Windows Multimedia Programmer's Reference*, the *Microsoft Windows for Pen Computing Programmer's Reference*, and the *Microsoft Windows Setup Toolkit*.

2.1 Changes in the Software Development Kit

The SDK offers the following enhancements:

- New dialog box editor (DLGEDIT.EXE)
- Information about new debugging and reporting tool (DRWATSON.EXE)
- New monitoring application for dynamic data exchange (DDESPY.EXE)
- Improved heap-walk program (HEAPWALK.EXE)
- New bitmap, icon, and cursor editor (IMAGEDIT.EXE)
- Multiple-resolution bitmap support for help files (MRBC.EXE)

- Segmented hypergraphics support for help files (SHED.EXE)
- New registration database updating tool (REGLOAD.EXE)
- New system debugging tool (DBWIN.EXE)
- New system resource stress-testing application (STRESS.EXE)
- Improved Microsoft Help Compiler (HC31.EXE)

The following sections describe changes to the SDK tools.

2.1.1 Dialog Editor (DLGEDIT.EXE)

Microsoft Dialog Editor has been completely rewritten for Windows 3.1. The new editor contains its own online reference information in Help.

2.1.2 Dr. Watson (DRWATSON.EXE)

Microsoft Windows Dr. Watson traps and reports general protection (GP) faults. You use Dr. Watson while testing and debugging your Windows application to gather information about GP faults your application causes. Dr. Watson creates a log file containing such information as the name of the faulting application or dynamic-link library (DLL), the instruction that caused the fault, and the stack trace leading up to the fault. If Dr. Watson finds a symbol (.SYM) file corresponding to the application or DLL, it displays the names of segments, functions, and variables in the stack trace. Dr. Watson is provided with the retail version of Windows 3.1. For information about how to read Dr. Watson log files, see *Microsoft Windows Programming Tools.*

2.1.3 DDESpy (DDESPY.EXE)

Microsoft Windows DDESpy lets you view information about dynamic data exchange (DDE) activity in your Windows system.

2.1.4 Heap Walker (HEAPWALK.EXE)

Microsoft Windows Heap Walker now runs only in protected mode (standard or 386 enhanced mode). Heap Walker displays segment names from symbol files for code segments and can display resources, such as icons, cursors, and menus.

2.1.5 Hotspot Editor (SHED.EXE)

Microsoft Windows Hotspot Editor creates segmented hypergraphics. A segmented hypergraphic is a graphic (bitmap or metafile) that has embedded hyperlinks that users choose to initiate some action, such as a jump to another Help topic. You use Hotspot Editor to create graphics to add to the help files for Windows Help. The editor contains its own online reference information in Help.

2.1.6 Image Editor (IMAGEDIT.EXE)

Microsoft Image Editor creates bitmaps, icons, and cursors that you can use as resources in your Windows applications. The new editor contains its own online reference information in Help.

2.1.7 Multiple-Resolution Bitmap Compiler (MRBC.EXE)

Microsoft Multiple-Resolution Bitmap Compiler creates bitmap files that contain several bitmaps representing the same image but having different resolutions, aspect ratios, or color formats. You use the compiler to create bitmap resources to add to help files for Windows Help. When a help file contains a multiple-resolution bitmap, Windows Help displays only the bitmap that best matches the resolution, aspect ratio, and color format of the current display device.

2.1.8 Registration File Loader (REGLOAD.EXE)

Microsoft Windows Registration File Loader adds entries to the registration database (REG.DAT) by reading a description of the entries from a text file and merging the information into the binary-format database. You can distribute REGLOAD.EXE with SHELL.DLL and the text file that describes your database entries if your application will be installed on Windows 3.0. If your application will be installed only on Windows 3.1, you need not ship REGLOAD.EXE, because the same functionality is built into Microsoft Windows Registration Editor (REGEDIT.EXE), which is distributed with the retail version of Windows 3.1.

2.1.9 System Debugging Log Application (DBWIN.EXE)

System Debugging Log Application allows you to display messages produced by the debugging version of Windows, even if you are not running a debugger and if you do not have a debugging terminal. It even makes it possible for you to see some errors when it is running with the retail version of Windows. DBWIN.EXE

allows you to control the output of specific varieties of messages. It also includes a feature that makes it possible for you to force memory-allocation errors, as an aid in testing the robustness of your application.

For information about the options you can set by using System Debugging Log Application, see Chapter 3, "Creating Windows Applications."

2.1.10 Stress-Resource Stress Application (STRESS.EXE)

Microsoft Stress-Resource Stress Application makes it possible for you to allocate system resources to see how your application behaves under low-resource conditions. You can allocate resources such as global memory, USER and graphics device interface (GDI) heap memory, file handles, and disk space.

2.1.11 Help Compiler (HC31.EXE)

Microsoft Help Compiler, version 3.1, supports metafiles, tables, segmented hypergraphics, secondary help windows, macros, and many other enhancements. The compiler creates help files that can be used with Microsoft Windows Help, version 3.1.

To support existing applications, the SDK also includes Microsoft Help Compiler, version 3.0 (HC30.EXE).

2.2 Changes in Windows Application Programming Interface

Windows 3.1 includes many new functions as well as enhancements to existing Windows functions. Windows has also added many new styles and messages to existing user controls. Following are some of the additions and enhancements:

- Drag-drop feature (SHELL.DLL)
- Registration database (SHELL.DLL)
- Dynamic Data Exchange Management Library (DDEML.DLL)
- Common dialog boxes (COMMDLG.DLL and international versions of the library)
- Object linking and embedding (OLECLI.DLL, OLESVR.DLL, SHELL.DLL)

- Data decompression (LZEXPAND.DLL)
- System resources stress-testing (STRESS.DLL)
- File installation and version checking (VER.DLL and VER.LIB)
- Tool help (TOOLHELP.DLL)
- Windows Help (WINHELP.EXE and WINHELP.HLP)
- TrueType fonts
- New printer functions
- Device-independent bitmap driver (DIB.DRV)
- Installable drivers
- Enhanced messages, styles, and functions

Most of the new Windows 3.1 functions are included in several new DLLs. The appropriate DLL must be present for a program to use the new functions.

For more information about any of these new features, see the *Microsoft Windows Programmer's Reference.*

2.2.1 Drag-Drop Feature

When an application implements the drag-drop feature, a user can select one or more files in Windows File Manager, drag them to an open application, and drop them there. The application receives a message it can use to retrieve the filenames and the coordinates of the point at which the files were dropped.

Following are the new drag-drop functions:

Function	Description
DragAcceptFiles	Registers whether a window accepts dropped files.
DragFinish	Releases memory allocated for dropping files.
DragQueryFile	Retrieves the filename of a dropped file.
DragQueryPoint	Retrieves the cursor position when a file is dropped.

The drag-drop feature depends on the dynamic-link library SHELL.DLL. This is a redistributable library.

2.2.2 Registration Database

The registration database is a systemwide source of information about applications. This information is used to support the integration of applications with File Manager for Windows 3.1 and is used by applications that support object linking and embedding (OLE).

Your application can add entries to the registration database by using the registration functions; you can also add entries when you install your application by using Microsoft Windows Registration Editor (REGEDIT.EXE).

Following are the new registration database functions:

Function	Description
RegCloseKey	Closes a key.
RegCreateKey	Creates a key.
RegDeleteKey	Deletes a key.
RegEnumKey	Enumerates subkeys of a specified key.
RegOpenKey	Opens a key.
RegQueryValue	Retrieves a text string for a specified key.
RegSetValue	Associates a text string with a specified key.

The registration database feature depends on the dynamic-link library SHELL.DLL. This is a redistributable library.

2.2.3 Dynamic Data Exchange Management Library

The Dynamic Data Exchange Management Library (DDEML) provides a set of API elements that simplifies the task of adding DDE capability to a Windows application. Instead of sending, posting, and processing DDE messages directly, an application uses the functions provided by the DDEML to manage DDE conversations.

The DDEML also provides a facility for managing the strings and data that are passed among DDE applications. Applications create and exchange string handles and data handles instead of atoms and pointers to shared memory objects. A server-name service allows a server application to register the application names that it supports. The names are broadcast to other applications in the system, which can

then use the names to connect to the server. The DDEML also ensures compatibility among DDE applications by forcing them to implement the DDE protocol in a consistent manner.

Following are the new DDE functions:

Function	Description
DdeAbandonTransaction	Abandons an asynchronous transaction.
DdeAccessData	Accesses a DDE global memory object.
DdeAddData	Adds data to a DDE global memory object.
DdeClientTransaction	Begins a DDE data transaction.
DdeCmpStringHandles	Compares two DDE string handles.
DdeConnect	Establishes a conversation with a server.
DdeConnectList	Establishes multiple DDE conversations.
DdeCreateDataHandle	Creates a DDE data handle.
DdeCreateStringHandle	Creates a DDE string handle.
DdeDisconnect	Terminates a DDE conversation.
DdeDisconnectList	Destroys a DDE conversation list.
DdeEnableCallback	Enables or disables one or more DDE conversations.
DdeFreeDataHandle	Frees a global memory object.
DdeFreeStringHandle	Frees a DDE string handle.
DdeGetData	Copies data from a global memory object to a buffer.
DdeGetLastError	Returns an error value set by a DDEML function.
DdeInitialize	Registers an application with the DDEML.
DdeKeepStringHandle	Increments the use count for a string handle.
DdeNameService	Registers or unregisters a server application name.
DdePostAdvise	Prompts a server to send data to a client during an advise loop.
DdeQueryConvInfo	Retrieves information about a DDE conversation.
DdeQueryNextServer	Obtains the next handle in a conversation list.
DdeQueryString	Copies string-handle text to a buffer.
DdeSetUserHandle	Associates a user-defined handle with a transaction.
DdeUnaccessData	Frees a DDE global memory object.
DdeUninitialize	Frees an application's DDEML resources.

The DDEML feature depends on the dynamic-link library DDEML.DLL. This is a redistributable library.

2.2.4 Common Dialog Boxes

Common dialog boxes are dialog boxes that applications display by calling a single function rather than by creating a dialog box procedure and a resource file containing a dialog box template. Common dialog boxes can be used only in protected mode (standard or 386 enhanced mode).

In addition to simplifying the development of Windows applications, common dialog boxes assist users by providing a standard set of controls for performing certain operations (such as selecting colors). As Windows developers begin using common dialog boxes in their applications, users will find that once they master the use of common dialog boxes in one application, they can easily perform the same operations in other applications.

Following are the new functions for common dialog boxes:

Function	Description
ChooseColor	Creates a Color dialog box.
ChooseFont	Creates a Font dialog box.
FindText	Creates a Find dialog box.
GetFileTitle	Retrieves a filename.
GetOpenFileName	Creates an Open dialog box.
GetSaveFileName	Creates a Save As dialog box.
PrintDlg	Creates a Print dialog box.
ReplaceText	Creates a Replace dialog box.

The common dialog box feature depends on the COMMDLG.DLL dynamic-link library. This library and international versions of it are redistributable. Common dialog boxes can be used with Windows 3.0 but only in standard or 386 enhanced mode.

2.2.5 Object Linking and Embedding

An application that uses OLE can cooperate with other OLE applications to produce documents containing different kinds of data, all of which are easily manipulated by the user. The user editing such a document is able to improve the document by employing the best features of many different applications. An application that implements OLE gives its users the ability to move away from an application-centered view of computing, in which the tool used to complete a task is often a single application, and toward a document-centered view, in which users can employ as many tools as they choose to complete a job.

A single OLE document can contain many kinds of data in many different formats; such a document is called a compound document. A compound document uses the facilities of different OLE applications to manipulate the different kinds of data it displays. Any kind of data format can be incorporated into a compound document; with little or no extra code, OLE applications can even support data formats that have not yet been invented. The user working with a compound document does not need to know which data formats are compatible with one another or how to find and start the applications that created the data. Whenever a user chooses to work with part of a compound document, the application responsible for that part of the document starts automatically.

A compound document could be a brochure that included text, charts, ranges of cells in a spreadsheet, and illustrations. The user working with this brochure could automatically switch between the applications that produced its components. The information could be embedded in the document, or the document could contain links to certain information instead of containing the information itself. If the brochure used links, it would provide only minimal storage for the data to which it was linked, and it could be updated automatically whenever the linked data changed.

The OLE feature depends on the dynamic-link libraries OLECLI.DLL, OLESVR.DLL, and SHELL.DLL. These are redistributable libraries.

2.2.6 Data Decompression Functions

The data decompression functions let applications decompress files that were compressed by Microsoft File Compression Utility (COMPRESS.EXE). You can use these functions and utility to minimize the number of disks required to distribute and install your application.

Following are the new data decompression functions:

Function	Description
CopyLZFile	Copies a source file to a destination file, expanding it if the file was compressed.
GetExpandedName	Retrieves the original name of a compressed file.
LZClose	Closes a file that was opened by using the **LZOpenFile** function or the **OpenFile** function.
LZCopy	Copies a source file to a destination file, expanding it if the file was compressed (intended for single-file copy operations).
LZDone	Frees memory allocated by the **LZStart** function.
LZInit	Retrieves a file handle for a compressed file.

Function	Description
LZOpenFile	Opens a file.
LZRead	Reads from a compressed file.
LZSeek	Positions the file pointer within a compressed file.
LZStart	Allocates memory for multiple-file copy operations.

The data decompression feature depends on the dynamic-link library LZEXPAND.DLL. This is a redistributable library.

2.2.7 System Resources Stress-Testing Functions

The stress-testing functions artificially consume system resources. This makes it possible for you to see how your application behaves in low-resource conditions. This library was designed to make testing easier and more realistic.

Following are the new stress-testing functions:

Function	Description
AllocDiskSpace	Creates a file so that all but a specified amount of disk space is allocated.
AllocFileHandles	Creates files so that all but a specified number of file handles are allocated.
AllocGDIMem	Allocates all available memory in the graphics device interface (GDI) heap, down to a specified memory-object size.
AllocMem	Allocates all available memory, down to a specified memory-object size.
AllocUserMem	Allocates all available memory in the USER heap, down to a specified memory-object size.
FreeAllGDIMem	Frees memory allocated by the **AllocGDIMem** function.
FreeAllMem	Frees all memory allocated by the **AllocMem** function.
FreeAllUserMem	Frees memory allocated by the **AllocUserMem** function.
UnAllocDiskSpace	Deletes the file created by the **AllocDiskSpace** function.
UnAllocFileHandles	Frees all file handles allocated by the **AllocFileHandles** function.

The stress testing feature depends on the dynamic-link library STRESS.DLL. This is a redistributable library.

2.2.8 File Installation and Version Checking Functions

The file installation and version checking functions enable applications to install files based on version information and to examine currently installed files. The functions let applications examine a new version-information resource, created using the **VERSIONINFO** statement, to determine version information about a Windows executable file. Installation programs use this resource to determine whether existing executable files should be updated.

Following are the new file installation and version checking functions:

Function	Description
GetFileResource	Extracts the resource located by the **GetFileResource-Size** function.
GetFileResourceSize	Determines whether a file contains a resource of a specified type and identifier.
GetFileVersionInfo	Returns a structure with file version information from the specified version information resource.
GetFileVersionInfoSize	Determines whether file version information is available and the size of a buffer to hold the information.
GetSystemDir	Returns the name of the current system directory.
GetWindowsDir	Returns the name of the current Windows directory.
VerFindFile	Determines where to install a file, based on whether it locates another version of the file.
VerInstallFile	Installs the file (requires information from the **VerFind-File** function).
VerLanguageName	Returns a text representation of a binary Microsoft language identifier.
VerQueryValue	Returns version information from the specified version information resource.

File installation and version checking depend on the dynamic-link library VER.DLL. This is a redistributable library.

These functions can also be used in Microsoft MS-DOS applications by linking with the run-time library VER.LIB. The **GetWindowsDir** and **GetSystemDir** functions exist only in the run-time version of this library; Windows has its own versions of these functions.

2.2.9 Tool Helper Functions

The tool helper functions simplify the job of writing Windows-hosted debugging applications.

Following are the new tool helper functions:

Function	Description
ClassFirst	Retrieves information about the first class in the class list.
ClassNext	Retrieves information about the next class in the class list.
GlobalEntryHandle	Retrieves information about a global memory object.
GlobalEntryModule	Retrieves information about a specific memory object.
GlobalFirst	Retrieves information about the first global memory object.
GlobalHandleToSel	Converts a global handle to a selector.
GlobalInfo	Retrieves information about the global heap.
GlobalNext	Retrieves information about the next global memory object.
InterruptRegister	Installs a function to handle system interrupts.
InterruptUnRegister	Removes the function that processed system interrupts.
LocalFirst	Retrieves information about the first local memory object.
LocalInfo	Fills a structure with information about the local heap.
LocalNext	Retrieves information about the next local memory object.
MemManInfo	Retrieves information about the memory manager.
MemoryRead	Reads memory from an arbitrary global heap object.
MemoryWrite	Writes memory to an arbitrary global heap object.
ModuleFindHandle	Retrieves information about a module.
ModuleFindName	Retrieves information about a module.
ModuleFirst	Retrieves information about the first module.
ModuleNext	Retrieves information about the next module.
NotifyRegister	Installs a notification callback function.
NotifyUnRegister	Removes a notification callback function.
StackTraceCSIPFirst	Retrieves information about a stack frame.
StackTraceFirst	Retrieves information about the first stack frame.
StackTraceNext	Retrieves information about the next stack frame.
SystemHeapInfo	Retrieves information about the USER heap.
TaskFindHandle	Retrieves information about a task.

Function	Description
TaskFirst	Retrieves information about the first task in the task queue.
TaskGetCSIP	Returns the next CS:IP value of a task.
TaskNext	Retrieves information about the next task in the task queue.
TaskSetCSIP	Sets the CS:IP of a sleeping task.
TaskSwitch	Switches to a specific address within a new task.
TerminateApp	Terminates an application.
TimerCount	Retrieves execution times.

The tool helper feature depends on the dynamic-link library TOOLHELP.DLL. The DLL requires a modified version of the virtual timer device VTD.386 to provide additional support for the **TimerCount** function. Applications that use this function must install the virtual timer device by adding an appropriate device= entry in the [386Enh] section of the SYSTEM.INI file. Both TOOLHELP.DLL and VTD.386 are redistributable files.

2.2.10 Windows Help

Windows Help, version 3.1, lets applications control and display online information about the application. Users can request Help by choosing commands from the menu or by pressing the F1 key. Your application monitors these requests and activates Windows Help to provide general or context-sensitive help.

The Windows Help features depend on the Windows Help application WINHELP.EXE. This application and the corresponding help file WINHELP.HLP are redistributable files.

2.2.11 TrueType Fonts

Windows 3.1 includes a new font technology called TrueType and at least 13 core TrueType fonts. In addition to the TrueType fonts that are included with Windows 3.1, many additional TrueType fonts are already available that users can purchase. TrueType fonts can be scaled and rotated; they allow the same fonts to be used on the screen as are used on printers; and they allow documents to be portable between printers, applications, and systems.

The TrueType font technology offers many benefits to application designers, at little or no cost. It is not necessary to revise an application written for Windows

3.0 in order for that application to use TrueType fonts. If an application needs to take full advantage of the greater precision and versatility available with TrueType fonts, however, it can use the following new font functions:

Function	Description
CreateScalableFontResource	Creates a resource file for a specified TrueType font.
EnumFontsFamilies	Retrieves the fonts available on a specified device.
GetCharABCWidths	Retrieves the ABC widths of consecutive TrueType characters.
GetFontData	Retrieves font-metric data from a TrueType font file.
GetGlyphOutline	Retrieves data describing the curves of a character in a TrueType font.
GetKerningPairs	Retrieves kerning pairs for the current font.
GetOutlineTextMetrics	Retrieves metrics defining TrueType fonts.
GetRasterizerCaps	Determines whether TrueType is installed.

The TrueType feature is an integral part of Windows and is available with Windows 3.1 only.

2.2.12 New Printer Functions

The printer functions simplify printing by replacing the cumbersome printer escapes used in Windows 3.0.

Following are the new printer functions:

Function	Description
AbortDoc	Terminates a print job.
EndDoc	Ends a print job.
EndPage	Ends a page.
ResetDC	Updates a device context.
SetAbortProc	Sets the abort function for a print job.
SpoolFile	Puts a file in the spooler queue.
StartDoc	Starts a print job.
StartPage	Prepares the printer driver to receive data.

The printer functions are an integral part of Windows and are available with Windows 3.1 only.

2.2.13 Device-Independent Bitmap Driver

The device-independent bitmap (DIB) driver lets applications create and manipulate device-independent bitmaps. The driver is a redistributable file.

2.2.14 Installable Drivers

An installable driver is a Windows DLL that a Windows application (or another Windows DLL) can open, enable, query, disable, and close. An application can perform these operations by calling the following functions:

Function	Description
CloseDriver	Closes an installable driver.
DefDriverProc	Calls the default installable-driver procedure.
GetDriverInfo	Retrieves installable-driver data.
GetDriverModuleHandle	Retrieves an installable driver's module handle.
GetNextDriver	Enumerates installed drivers.
OpenDriver	Opens an installable driver.
SendDriverMessage	Sends a message to an installable driver.

2.2.15 New Messages

The following new messages have been added to Windows 3.1:

Message	Description
CB_FINDSTRINGEXACT	Finds a string in the list box of a combo box.
CB_GETDROPPEDCONTROLRECT	Retrieves the rectangle of the drop-down list box of a combo box.
CB_GETDROPPEDSTATE	Determines whether the list box of a combo box is visible.
CB_GETEXTENDEDUI	Determines whether a combo box has the extended interface.
CB_GETITEMHEIGHT	Retrieves the height of items in a combo box.
CB_SETEXTENDEDUI	Sets the default or extended user interface.
CB_SETITEMHEIGHT	Sets the height of items in a combo box.

Message	Description
EM_GETFIRSTVISIBLELINE	Retrieves the index of the top line in a multiline edit control.
EM_GETPASSWORDCHAR	Retrieves the password character displayed in an edit control.
EM_GETWORDBREAKPROC	Retrieves the wordwrap function for an edit control.
EM_SETREADONLY	Sets the read-only state of an edit control.
EM_SETWORDBREAKPROC	Provides custom word breaks in edit controls.
LB_FINDSTRINGEXACT	Finds a string in a list box.
LB_GETCARETINDEX	Retrieves the index of the list box item with the focus rectangle.
LB_GETITEMHEIGHT	Retrieves the height of items in a list box.
LB_SETCARETINDEX	Sets the focus rectangle in a list box.
LB_SETITEMHEIGHT	Sets the height of items in a list box.
STM_GETICON	Retrieves the icon handle associated with an icon control.
STM_SETICON	Associates an icon handle with an icon control.
WM_CHOOSEFONT_GETLOGFONT	Retrieves a **LOGFONT** structure for the Font common dialog box.
WM_COMMNOTIFY	Notifies a window about the status of its queues.
WM_DROPFILES	Indicates that a file has been dropped.
WM_PALETTEISCHANGING	Informs applications that an application is changing its palette.
WM_POWER	Indicates that the system is entering suspended mode.
WM_QUEUESYNC	Delimits computer-based training (CBT) messages.
WM_SYSTEMERROR	Indicates that a system error has occurred.
WM_WINDOWPOSCHANGED	Notifies a window that its size or position has changed.
WM_WINDOWPOSCHANGING	Notifies a window that its size or position is changing.

2.2.16 New Control and Window Styles

The following new user control and window styles have been added to Windows 3.1:

Style	Description
CBS_DISABLENOSCROLL	Shows a disabled vertical scroll bar in the combo box when the box does not contain enough items to scroll.
ES_READONLY	Prevents the user from typing or editing text in the edit control.
ES_WANTRETURN	Specifies that a carriage return should be inserted when the user presses the ENTER key while entering text into a multiline edit control in a dialog box.
LBS_DISABLENOSCROLL	Shows a disabled vertical scroll bar in the list box when the box does not contain enough items to scroll.
WS_EX_ACCEPTFILES	Specifies that a window created with this style processes the WM_DROPFILES message. This style is used with the **CreateWindowEx** function.
WS_EX_TOPMOST	Places a window above all non-topmost windows and keeps it above them even when the window is deactivated. This style is used with the **CreateWindowEx** function.
WS_EX_TRANSPARENT	Specifies that a window created with this style is transparent. This style is used with the **CreateWindowEx** function.

2.2.17 New Graphics Device Interface (GDI) Functions

The following new GDI functions have been added to Windows 3.1. This table does not include the new printer and TrueType functions, which are also part of GDI. (The new printer and TrueType functions are listed earlier in this chapter.)

Function	Description
GetAspectRatioFilterEx	Retrieves the current aspect-ratio filter.
GetBoundsRect	Retrieves the current accumulated bounding rectangle.
GetBrushOrgEx	Retrieves the origin of the current brush.

Function	Description
GetCurrentPositionEx	Retrieves the current position, in logical units, putting the result in a **POINT** structure.
GetTextExtentPoint	Retrieves the dimensions of a string.
GetViewportExtEx	Retrieves the viewport extent, putting the result in a **SIZE** structure.
GetViewportOrgEx	Retrieves the viewport origin, putting the result in a **POINT** structure.
GetWindowExtEx	Retrieves the window extent, putting the result in a **SIZE** structure.
GetWindowOrgEx	Retrieves the window origin, putting the result in a **POINT** structure.
IsGDIObject	Determines if the handle is not a handle of a GDI object.
MoveToEx	Moves the current position, putting the previous position in a **POINT** structure.
OffsetViewportOrgEx	Moves the viewport origin, putting the previous origin in a **POINT** structure.
OffsetWindowOrgEx	Moves the window origin, putting the previous origin in a **POINT** structure.
ResetDC	Updates a device context.
ScaleViewportExtEx	Scales viewport extents, putting the previous extents in a **SIZE** structure.
ScaleWindowExtEx	Scales window extents, putting the previous extents in a **SIZE** structure.
SetBitmapDimensionEx	Sets the width and height of a bitmap, putting the previous dimensions in a **SIZE** structure.
SetBoundsRect	Controls bounding-rectangle accumulation.
SetMetaFileBitsBetter	Create a memory object from a metafile.
SetViewportExtEx	Sets viewport extents, putting the previous extents in a **SIZE** structure.
SetViewportOrgEx	Sets the viewport origin, putting the previous origin in a **POINT** structure.
SetWindowExtEx	Sets window extents, putting the previous extents in a **SIZE** structure.
SetWindowOrgEx	Sets the window origin, putting the previous origin in a **POINT** structure.

2.2.18 New KERNEL Functions

The following new KERNEL functions have been added to Windows 3.1:

Function	Description
DebugOutput	Sends formatted messages to the debugging terminal.
DirectedYield	Forces execution to continue at a specified task.
GetSelectorBase	Retrieves the base address of a selector.
GetSelectorLimit	Retrieves the limit of a selector.
GetWinDebugInfo	Queries current system-debugging information.
hmemcpy	Copies bytes.
_hread	Reads from a file.
_hwrite	Writes to a file.
IsBadCodePtr	Determines whether a code pointer is valid.
IsBadHugeReadPtr	Determines whether a huge read pointer is valid.
IsBadHugeWritePtr	Determines whether a huge write pointer is valid.
IsBadReadPtr	Determines whether a read pointer is valid.
IsBadStringPtr	Determines whether a string pointer is valid.
IsBadWritePtr	Determines whether a write pointer is valid.
IsDBCSLeadByte	Determines whether a character is the lead byte, the first byte of a character in a double-byte character set (DBCS).
IsTask	Determines whether a task handle is valid.
LogError	Identifies an error message.
LogParamError	Identifies a parameter-validation error.
SetSelectorBase	Sets the base of an existing selector.
SetSelectorLimit	Sets the limit of a selector.
SetWinDebugInfo	Sets current system-debugging information.

2.2.19 New USER Functions

The following new USER functions have been added to Windows 3.1:

Function	Description
CallNextHookEx	Passes hook information down the hook chain.
CopyCursor	Copies a cursor.

Function	Description
CopyIcon	Copies an icon.
EnableCommNotification	Enables or disables WM_COMMNOTIFY posting to window.
EnableScrollBar	Enables or disables scroll-bar arrows.
GetClipCursor	Retrieves cursor-confining rectangle coordinates.
GetCursor	Returns the current cursor handle.
GetDCEx	Retrieves the handle of a device context.
GetFreeSystemResources	Returns the percentage of free system resource space.
GetMessageExtraInfo	Retrieves information about a hardware message.
GetOpenClipboardWindow	Returns a handle of the window that currently has the clipboard open.
GetQueueStatus	Determines the queued message type.
GetSystemDebugState	Returns system-state information to a debugger.
GetTimerResolution	Retrieves the timer resolution.
GetWindowPlacement	Retrieves the show state and the normal (restored), minimized, and maximized positions of a window.
hardware_event	Places a hardware message in the system queue.
IsMenu	Determines whether a menu handle is valid.
LockInput	Locks input to all tasks except the current one.
LockWindowUpdate	Disables or reenables drawing in a window.
MapWindowPoints	Converts points to another coordinate system.
QuerySendMessage	Determines whether a message originated within a task.
RedrawWindow	Updates a client rectangle or region.
ScrollWindowEx	Scrolls a window's client area.
SetWindowPlacement	Sets the show state and the normal (restored), minimized, and maximized positions of a window.
SetWindowsHookEx	Installs a hook function into a hook chain.
SubtractRect	Creates a rectangle from the difference between two rectangles.
SystemParametersInfo	Queries or sets systemwide parameters.
UnhookWindowsHookEx	Removes a function from the hook chain.
WNetAddConnection	Adds network connections.
WNetCancelConnection	Removes network connections.
WNetGetConnection	Lists network connections.

2.3 Compatibility Issues

Although every effort has been made to ensure that the many enhancements and improvements to Windows are compatible with Windows 3.0 applications, some enhancements may affect application operation. This is especially true if an application uses features in an undocumented fashion or relies on invalid assumptions about the behavior of Windows. For a complete discussion of compatibility issues, see the Compatibility Issues topic in either Microsoft Windows Help or Microsoft QuickHelp (QH.EXE).

Creating Windows Applications

Chapter 3

This chapter explains what elements are needed to build applications for the Microsoft Windows operating system versions 3.0 and 3.1. It also provides guidelines for writing robust applications and for debugging applications.

3.1 Writing Compatible Windows Applications

The Microsoft Windows 3.1 Software Development Kit (SDK) allows you to create applications for either Windows 3.0 or 3.1. If you write your application carefully, you can create a single application that is compatible with Windows 3.0 but also takes advantage of newer features when running with Windows 3.1.

3.1.1 Windows 3.1 Applications

The Windows 3.1 SDK tools, header files, and libraries create Windows 3.1 applications by default. No special procedures are required to create executable files that run with Windows 3.1. If you create Windows Help files for your applications, use Microsoft Help Compiler version 3.1 (HC31.EXE) to compile your files so that they have access to the latest features of Windows Help.

Applications that call Windows 3.1 functions depend on Windows 3.1 and cannot be run with Windows 3.0.

3.1.2 Windows 3.0 Applications

You can use the Windows 3.1 SDK to create a Windows 3.0 application by following these steps:

1. Set the WINVER define variable to 0x300 to enable the WINDOWS.H file for Windows 3.0 compilation. Place the following statement immediately before the include statement for the header file:

   ```
   #define WINVER 0x300
   ```

2. Link your application object files with the LIBW.LIB library provided with the Windows 3.1 SDK. Except for the functions that are new to Windows 3.1, all functions defined in this import library are compatible with Windows 3.0.
3. Mark your application as a Windows 3.0-only executable by using the **/30** option with Windows Resource Compiler (RC). The **/30** option cannot be used with the **/r** option.
4. If you create Windows Help files for your application, use Help Compiler version 3.0 (HC30.EXE) to compile your files.

By default, Resource Compiler marks applications for 3.1, so it is important to use the **/30** option mentioned in the preceding steps.

All Windows 3.0 applications can use Windows extensions, such as common dialog boxes and object linking and embedding. If you use these features, you must ship the corresponding dynamic-link libraries (DLLs) and related files with your application. They should be installed along with the application.

3.1.3 Combined Windows 3.0 and 3.1 Applications

You can create Windows applications that run with Windows 3.0 but also take advantage of newer features when running with Windows 3.1. Such applications consist primarily of Windows 3.0 function calls but conditionally link to and use Windows 3.1 functions.

To build a combined application, mark your application as a Windows 3.0 only executable by using the **/30** option with Resource Compiler, but do not set the WINVER define variable to 0x300. You must use the **GetVersion** function to determine the version of Windows that is running before using any Windows 3.1 functions.

The following example demonstrates how to set a flag if the current system is Windows 3.1:

```
extern BOOL fWin31;

UINT version;

fWin31 = FALSE;

version = LOWORD(GetVersion());
if (((LOBYTE(version) << 8) | HIBYTE(version)) >= 0x030a) {
    fWin31 = TRUE;
}
```

For information about interpreting the return value of the **GetVersion** function, see the *Microsoft Windows Programmer's Reference, Volume 2.*

Your application can call Windows 3.1 functions directly as long as you link it with the 3.1 version of LIBW.LIB. (It is not necessary to call the **GetProcAddress** function.) However, you must ensure that Windows 3.1 functions are not called when your application is running with Windows 3.0. The following example demonstrates how this can be done, using the fWin31 flag that was set in the preceding example:

```
extern BOOL fWin31;

if (fWin31) {
    ScrollWindowEx(hwnd, ...);  /* new for Windows 3.1  */
}
else {
    ScrollWindow(hwnd, ...);    /* Windows 3.0 function */
}
```

If you create Windows Help files for your application, either use Help Compiler version 3.0 (HC30.EXE) to compile your files, or create a help file for Windows 3.1 and release your help file with the redistributable Windows 3.1 versions of the WINHELP.EXE and WINHELP.HLP files.

If you run a combined application using the debugging version of Windows 3.0, a call to an undefined function causes a warning. The application, however, continues to load and run successfully, as long as the function is not actually called.

3.2 Creating Robust Applications

Windows 3.1 includes a number of features and enhancements designed to make running Windows applications much more reliable. Efforts to make Windows 3.1 more reliable have focused primarily on three areas:

- Improving how the system handles errors if and when they occur.
- Avoiding errors in system code by ensuring the validity of all handles, pointers, structures, indices, and flags passed to the system.
- Providing better diagnostics, tools, and header files for finding and fixing bugs more efficiently during development.

The two key components improving reliability are the parameter validation built into the Windows operating system and the STRICT type-checking of the WINDOWS.H file. Also useful are the new features of WINDOWSX.H, which include macros, message crackers, and control functions.

3.2.1 Parameter Validation

Windows 3.1 contains code to validate parameters passed to Windows functions and messages. These features are included in both the retail and debugging versions of the system. The debugging version of the system includes some additional features and parameter checking that is not included in the retail product.

3.2.1.1 Invalid Parameter Error Messages

The system validates handles, pointers, structures, indices, and flags. In most cases, an invalid parameter causes a function to return an error value. In other cases, such as when a flag is invalid, the function executes as usual, but an appropriate warning message is displayed.

When Windows encounters an invalid parameter error, it displays the message on your debugging terminal or window. The message has the following form:

err *AppName function*:*address*: *message*:*parameter-value*

Following are the message parameters:

AppName
Identifies the application or DLL that caused the error.

function
Identifies the number of the function that was passed the invalid parameter.

address
Identifies the address of the function that was passed the invalid parameter.

message
Specifies the string identifying the error.

parameter-value
Specifies the value of the invalid parameter.

For example, a message could have the following form:

```
err FONTSAMP 011F:056A: Invalid local handle: 1D50
```

If the address is not near the address of a Windows function you recognize, the window message parameter is probably invalid. Functions that take messages, such as **SendMessage**, **DispatchMessage** and **SendDlgItemMsg**, show an address within the message validation code. Parameter values for invalid parameters begin with the PV prefix (for example, PV_WM_COMMAND).

By default, invalid parameter messages display a stack trace and an "Abort, Break, or Ignore?" prompt. You can change the default by setting options in the System Debug Options box of the Systems Debugging Log Application (DBWIN.EXE). This dialog box is displayed when you choose the Settings command on the Options menu. (For more information about DBWIN.EXE, see Section 3.3.2, "Using the System Debugging Log Application.")

You can also log invalid parameter errors by using Dr. Watson, just as you would log general-protection (GP) faults by using Dr. Watson. By default, this feature is turned off. For more information on invalid parameter error logging, see *Microsoft Windows Programming Tools*.

3.2.1.2 Buffer Overflow Errors

A common application error is to allocate too little space for a buffer that is passed to and filled by Windows. These errors are especially difficult to track if the buffers are allocated on the stack. Windows can help you find these errors by filling

buffers before information is copied into them. If the operation overflows the buffer, Windows detects and reports the error.

By default, this feature is disabled. You can enable the feature by choosing the Settings command on the Options menu of DBWIN.EXE and then selecting the Fill Buffers check box. When you select this check box, Windows displays a stack trace and an "Abort, Break, or Ignore?" prompt with some warning messages.

This feature is available in the Windows debugging version only.

3.2.1.3 Interpreting Invalid Parameters

Possible reasons for getting invalid parameter errors follow.

Invalid Handles

A handle is invalid under the following circumstances:

- Using NULL or –1 when it is not allowed
- Reusing a destroyed or deleted handle
- Using an uninitialized stack variable
- Using a device context handle created by the **CreateIC** or **CreateMetaFile** function in a function that does not allow the handle
- Passing one type of handle in place of another, such as passing a device-context handle in place of an window handle

Invalid Pointers

A pointer is invalid under the following circumstances:

- Using a NULL pointer when it is not allowed
- Pointing to a buffer that is too small
- Using a function pointer without properly exporting it or properly creating a procedure-instance address
- Pointing to a string that does not have a null-terminating character
- Pointing to a structure that contains an invalid member (for example, if you call the **RegisterClass** function with a invalid window procedure in the **CREATESTRUCT** structure, Windows reports an invalid pointer)
- Using an uninitialized stack variable
- Passing a read-only pointer when a read-write pointer is required

Invalid Flags or Value

A flag or value is invalid under the following circumstances:

- Passing meaningless flags
- Passing an out-of-range index
- Using a value that is otherwise illegal

You can use pointer-validation functions (such as **IsBadCodePtr**) to help you check for and debug your application's use of pointers.

3.2.2 Strict Type-Checking

The Windows 3.1 header file (WINDOWS.H) includes various features for detecting problems when compiling an application. These features, which are provided by the STRICT option, make application development faster and easier.

You define STRICT before the include statement for the header file. STRICT causes the various types and function prototypes in WINDOWS.H to be declared with very strict type-checking. For example, once STRICT is defined, it is impossible to pass a window handle to a function that requires a device-context handle without generating a compiler error.

3.2.2.1 Features of the STRICT Option

Specific features provided by the STRICT option include:

- Strict handle type-checking
- Proper declaration of certain parameter and return value types
- Fully prototyped type definitions for callback function types
- Proper declaration of polymorphic parameters and return values (for example, *wParam* and *lParam* message parameters)
- Proper use of the **const** keyword for pointer parameters and structure members when the pointer is read-only

Type declarations for many of the Windows functions and callback functions have changed. Nonetheless, unless you define STRICT, the new declarations for Windows 3.1 are fully compatible with the old declarations for Windows 3.0. WINDOWS.H for Windows 3.1 can, therefore, be used to compile Windows 3.0 applications without modifications.

3.2.2.2 Compiling with the STRICT Option

In general, the STRICT option is most useful with newly developed code or with code that is being maintained or changed regularly. Code that has already been written and tested, and is not changing very much over time, will generally not benefit as much from STRICT. If you find that stable code generates lots of run-time parameter validation errors when run with Windows 3.1, you will find STRICT very valuable as you go through the code to clean up those errors.

The following procedures will ensure that your application conforms to STRICT type-checking:

- Use new handle and parameter types. In particular, replace **HANDLE** with appropriate handle types and use **WPARAM** and **LPARAM** with all message parameters.
- Use new return type and parameter types for windows and dialog box procedures and callback functions.
- Declare all your functions with full prototypes. Place these prototypes in a include file and include it with each source file.
- Cast function pointers to the proper type rather than to **FARPROC** type. This is especially important with the **MakeProcInstance** function.
- Take special care with **HMODULE** and **HINSTANCE** types. There are a few Windows functions that return or accept only **HMODULE** types.
- Use the **MAKELPARAM** macro instead of the **MAKELONG** macro when building **LPARAM** parameters out of two words. Also, use the **MAKELRESULT** macro instead of **MAKELONG** when building **LRESULT** return values.
- Cast the handle or near pointer to a **WORD** type in order to prevent getting the data segment value in the high word of the value when casting a handle or near pointer value to **LRESULT** or **LPARAM**.
- Cast a far pointer, **LPARAM** or **LRESULT**, to a **DWORD** type and then to the desired type when you cast the far pointer to a handle or near pointer. This prevents "segment lost in conversion" warnings.
- Make sure you have the following lines, in the given order, in each source file:

```
#define STRICT
#include <windows.h>
```

- Compile your source to use the highest level of error checking. Treat any warnings as errors and correct your sources to eliminate the warning messages.
- Link and run the application to ensure that it executes without errors.

3.3 Testing and Debugging Your Application in Windows

One of the advantages of an operating system such as Windows is its ability to run more than one application at a time. However, this advantage can also create hazards when you are testing and debugging an application.

Windows is a robust operating system. When Windows is running in protected (standard or 386 enhanced) mode, it can usually terminate an application that encounters a fatal error (such as an invalid handle) without affecting other applications. A fatal error or even a GP fault in an application very rarely causes the entire system to crash. However, it is possible to cause system failure in other ways when you are testing and debugging an application.

Because of the risk of system failure, you should always save all file buffers to disk before testing and debugging your application. You should also avoid running other applications while testing and debugging your application if a general system failure would cause problems for the other applications.

3.3.1 Using Different Windows Versions

The Windows 3.1 SDK provides two environments for debugging or testing your Windows applications: a debugging version of the retail Windows product and a nondebugging version of the retail Windows product.

The SDK installation program creates two directories to contain the debugging and nondebugging versions of the core DLLs. Unless you specify different paths, Install places the debugging versions of the Windows core libraries in the directory \WINDEV\DEBUG and the nondebugging versions in the directory \WINDEV\NODEBUG. Install copies the nondebugging version of the Windows core libraries from your Windows system directory to the \WINDEV\NODEBUG directory.

You can conveniently switch between the debugging and nondebugging versions of Windows by running one of two batch files that Install places in the Windows development directory (named \WINDEV by default). The N2D.BAT file

switches from the nondebugging to the debugging version, and D2N.BAT switches from the debugging to the nondebugging version.

These batch files either copy files from the directories \WINDEV\DEBUG and \WINDEV\NODEBUG or rename files in your Windows system directory. When you install the SDK files, Install asks if you want to keep a duplicate set of the libraries and symbol files in your Windows system directory. If you answer Yes, N2D.BAT and D2N.BAT quickly rename the duplicate files. Otherwise, the batch files copy the DLLs to your Windows system directory from the appropriate directory.

If you choose to retain a duplicate set of files, the DLLs and symbol files for the two versions of Windows appear in your Windows system directory with the same names as the core libraries and symbol files, but with the letter *N* (nondebugging) or *D* (debugging) appended to the name. For example, in addition to the GDI.EXE file, your system directory will contain the GDID.EXE and GDIN.EXE files.

3.3.1.1 Debugging Version

The debugging version of Windows consists of a set of DLLs that replace the Windows core DLLs of the retail product. The replaced DLLs are USER.EXE, KRNL286.EXE, KRNL386.EXE, GDI.EXE, and MMSYSTEM.DLL. Accompanying these DLLs is a set of symbol (.SYM) files.

The debugging versions of the core DLLs provide error checking and diagnostic messages that help you debug a Windows application. The symbol-file information helps you track calls into Windows when using the Microsoft Windows 80386 Debugger (WDEB386.EXE). In addition, the debugging versions of these DLLs contain Microsoft® CodeView® symbol information for tracking calls into Windows when using Microsoft® CodeView® for Windows™ (CVW).

A special setting is available in the [386Enh] section of SYSTEM.INI for the debugging version of Windows. The form of this setting follows:

```
DebugPhysAddrs = {TRUE|FALSE}
```

By default, Windows makes the entire base physical linear memory region available when a debugger is loaded. Setting the DebugPhysAddrs option to FALSE overrides this default when the debugger is loaded. Although the FALSE setting prevents you from being able to examine all memory, it creates a memory environment more like the nondebugging version of Windows, which can help you spot problems with pointers more quickly. The default value for DebugPhysAddrs is TRUE.

3.3.1.2 Nondebugging Version

During application development, you should use the debugging version of Windows. However, use the nondebugging version of Windows whenever you want to do the following:

- Test the final version of your application
- Test the performance of your application without the performance disadvantages of the debugging version of Windows

Use the nondebugging version of Windows with the core DLLs supplied by the retail version of Windows. The Windows 3.1 SDK also provides symbol files for the nondebugging version of Windows. The retail Windows core libraries do not contain CodeView symbol information, however.

3.3.2 Using the System Debugging Log Application

The System Debugging Log Application (DBWIN.EXE) allows you to display messages produced by the debugging version of Windows even if you are not running a debugger and do not have a debugging terminal. DBWIN.EXE allows you to control the output of specific types of messages. It also includes a feature that forces memory-allocation errors when testing the robustness of an application.

Note DBWIN.EXE can provide useful debugging messages with the retail version of Windows as well. When you run DBWIN.EXE with the retail version of Windows, the Settings and Alloc Break commands are disabled in the Options menu, and you will only see a limited subset of debugging messages.

3.3.2.1 System Debugging Output

The default system debugging output goes to AUX. DBWIN.EXE can also send debugging messages to COM1 or COM2. Sending debugging output to COM1 or COM2 improves the performance of your debugging system when you have redirected system debugging output to NUL, or if DBWIN.EXE is not running.

To disable AUX as the default, add the following setting to the [Debug] section of SYSTEM.INI:

```
OutputTo=NUL
```

To disable the default kernel output and to send output to COM1 or COM2, set the MS-DOS COM port baud rates to match the baud rates of your debugging terminal by using the MS-DOS **mode** command. To ensure that the settings are always correct, use the **mode** command in your AUTOEXEC.BAT file.

You can log messages to the system debugging window, to a monochrome screen, or to the COM1 or COM2 devices. The default destination for messages is to a window.

You can choose different destinations for debugging messages from the Options menu. These settings stay in effect the next time you run DBWIN.EXE.

3.3.2.2 System Debug Options Dialog Box

When you choose the Settings command on the Options menu, a System Debug Options dialog box appears. This dialog box allows you to control the output of debugging messages produced by the debugging version of Windows.

The System Debug Options dialog box works only when you are running the debugging version of Windows. There are three groups of check boxes, described as follows:

Break Options
Control whether and how a message will cause a break and stack trace to the debugger.

Debug Options
Control the kind of debugging features that are enabled in the system.

Trace Options
Control whether or not certain kinds of informational messages are produced.

The check boxes for Break Options and Trace Options are self-explanatory. The following list explains the check boxes for the Debug Options:

Option	Description
Validate Heap	Checks the consistency of global and local heaps before every call to a memory-management function. This option affects the global heap only when it is one of the default startup settings (that is, when it is saved by choosing the Save Settings command on the File menu). This option affects local heaps only if it is set before the application is started.

Option	Description
Check Free Blocks	Ensures that freed local blocks are not written into. The value 0xFB is written into free blocks, and when the heap is validated, a check is performed to ensure that the blocks are still filled with this value. This option works only with local heaps. This option must be used with the Validate Heap option.
Buffer Fill	Fills buffers that are passed to Windows functions with the value 0xF9. This option ensures that all of the supplied buffer is writable and helps detect overwrite problems that can occur when the buffer is too small.
Break with INT 3	Breaks to the debugger with an **int 3** instruction, instead of a fatal exit. This option does not display a stack trace.
Don't trap faults	Prevents the system from hooking GP and stack overflow faults. (Many faults that result from choosing this option would normally be handled by the system. Choosing this option results in faults that would not occur otherwise.)

3.3.2.3 Alloc Break Command

The Alloc Break command on the Options menu ensures that an application deals properly with out-of-memory conditions. This command displays a dialog box into which you can enter the module name of your application and the number of memory allocations you want to succeed before subsequent allocations fail.

The system counts each global or local memory allocation performed by your application. When the number of allocations reaches the allocation break count, that allocation and all subsequent allocations fail. Because memory allocations made by the system fail once the break count is reached, calls to certain functions (such as **CreateWindow**, **CreateBrush**, and **SelectObject**) fail as well. Only allocations made within the context of the application you specify are affected by the allocation break count.

The module name is limited to 8 characters. In some cases the module name may be different from the filename; the module name is specified in the module-definition (.DEF) file for the application. You cannot specify the module name of a DLL.

If you set the break count to zero, no allocation break is set, but the system counts allocations made by the specified application. You can choose the Show Count button to display the current allocation count.

You can set an allocation break before an application is run. The allocation count is then set to zero and allocations are counted as soon as the application starts. If you run more than one instance of an application, the allocation break applies only to the most recent instance.

The allocation count is also reset to zero when you choose the Set button or the Inc & Set button. You can set an allocation break before performing an operation to ensure that your application handles the problem effectively. Then you can choose Inc & Set and repeat the operation to ensure that the next allocation failure is also handled properly.

Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399

Microsoft®

0392 Part No. 28914