

Microsoft[®] Operating System/2

Programmer's Toolkit

Programming Tools

Version 1.0

Microsoft Corporation

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft Corporation. The software and/or databases described in this document are furnished under a license agreement or nondisclosure agreement. The software and/or databases may be used or copied only in accordance with the terms of the agreement. The purchaser may make one copy of the software for backup purposes. No part of this manual and/or database may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the purchaser's personal use, without the written permission of Microsoft Corporation.

© Copyright Microsoft Corporation, 1988. All rights reserved.
Simultaneously published in the U.S. and Canada.

Microsoft®, MS®, the Microsoft logo, and CodeView® are registered trademarks of Microsoft Corporation.

Document No. 060060003-100-R00-388
Part No. 01889

Contents

1 Introduction 1

- 1.1 Overview 3
- 1.2 Programmer's Toolkit Programs 3
- 1.3 Documentation 4
- 1.4 About This Guide 5
- 1.5 Notational Conventions 5

2 Building MS OS/2 Programs and Dynamic-Link Libraries 7

- 2.1 Introduction 9
- 2.2 Setting Up Your Development Environment 9
- 2.3 Building an MS OS/2 Program 9
- 2.4 Building a Dynamic-Link Library 15
- 2.5 Writing an MS OS/2 Assembly-Language Program 20
- 2.6 Writing an Assembly-Language Dynamic-Link Library 22

3 Miscellaneous Development Tools 27

- 3.1 Introduction 29
- 3.2 The Kbdp Program 30
- 3.3 The Libwhere Program 31
- 3.4 The Mkmsgf Program 32
- 3.5 The Msgbind Program 34
- 3.6 The Shd Program 36
- 3.7 The Where Program 39

A Using the QuickHelp Program 41

- A.1 Introduction 43
- A.2 The MS OS/2 System Functions Database 43
- A.3 Installing QuickHelp 44
- A.4 Starting QuickHelp 45
- A.5 Quitting QuickHelp 46
- A.6 Getting Help with QuickHelp 46
- A.7 Choosing Commands in QuickHelp Menus 46
- A.8 Choosing a Category 47

A.9	Choosing a Topic from a Topic List	48
A.10	Viewing a Topic	49
A.11	Choosing a Topic from the QuickHelp Window	49
A.12	Choosing a Topic from the Screen	50
A.13	Searching for a Topic	51
A.14	Viewing Related Topics	52
A.15	Searching Within a Topic	54
A.16	Pasting from the Database	55
A.17	Opening a New Database	56
A.18	Using QuickHelp with the Microsoft Editor	57
A.19	Specifying Parameters on the Command Line	59
A.20	Using the QH Environment Variable	61

Index	63
--------------	-----------

Figures

Figure 2.1	The Stack Frame	23
Figure A.1	QuickHelp Window	45
Figure A.2	Categories Menu	47
Figure A.3	Kbd Topic List	48
Figure A.4	KbdCharIn References Menu	52
Figure A.5	KbdPeek Window	53

1

2

3

Chapter 1

Introduction

1.1	Overview	3
1.2	Programmer's Toolkit Programs	3
1.3	Documentation	4
1.4	About This Guide	5
1.5	Notational Conventions	5

1.1 Overview

The Microsoft® Operating System/2 Programmer's Toolkit provides you with a set of programming tools for writing real-mode, protected-mode, or dual-mode MS® OS/2 applications. This toolkit, along with a compiler or the Microsoft Macro Assembler, a debugger, and a text editor, is a complete software development system for MS OS/2 applications. You can use several compilers for MS OS/2 program development, including the Microsoft C Optimizing Compiler, Microsoft BASIC Compiler, Microsoft FORTRAN Compiler, Microsoft Pascal Compiler, or another vendor's MS OS/2 compiler.

1.2 Programmer's Toolkit Programs

The toolkit contains the following programs:

Program	Description
bind	Converts protected-mode programs into dual-mode programs.
exehdr	Displays the contents of an MS OS/2 executable file header.
implib	Creates an import library.
kbdp	Sets the rate of keyboard typing.
lib	Creates, combines, and maintains run-time libraries.
libwhere	Searches for and displays the location of a library.
link	Links one or more object files with libraries and module-definition files to produce an MS OS/2 application.
mkmsgf	Creates a message file for use with MS OS/2 message functions.
msgbind	Binds a message segment to a program.
QuickHelp	Displays help information about MS OS/2 functions and topics.
shd	Displays the contents of a file in hexadecimal or other notation.
where	Searches for and displays the location of a file.

In addition to these programs, the MS OS/2 Programmer's Toolkit provides the **trace** and **tracefmt** programs. You will find descriptions of these programs in the *Microsoft Operating System/2 User's Reference*.

The MS OS/2 program examples supplied illustrate how specific MS OS/2 functions are used. Several games and sample programs are also provided to show how MS OS/2 functions can be used to create an application.

1.3 Documentation

In addition to the MS OS/2 Programmer's Toolkit documentation, you will see the following Microsoft manuals and products referred to throughout this manual. They contain information about other helpful programming tools.

- The Microsoft CodeView® and Utilities manual describes the CodeView debugger and the **errout**, **exemod**, **exepack**, **lib**, **link**, **make**, and **setenv** programs.
- *Microsoft CodeView and Utilities Update*—describes the MS OS/2-specific features of the CodeView debugger and the **bind**, **exehdr**, **ilink**, **implib**, **link**, and **make** programs. This manual, which supplements the Microsoft CodeView and Utilities manual, provides additional information about MS OS/2 program development. Of special interest to programmers are the sections that describe how to debug dynamic-link modules and multiple-thread programs, and how to create import libraries and module-definition files.
- *Microsoft C Optimizing Compiler Version 5.1 Update*—describes features of the Microsoft C Optimizing Compiler version 5.1 that support MS OS/2 programming.
- *Microsoft Mixed-Language Programming Guide*—describes the concepts, syntax, and programming methods for creating a mixed-language program. The first part of the manual shows how to establish an interface between two languages and the second part shows how to pass different kinds of data between languages.

1.4 About This Guide

The *Microsoft Operating System/2 Programming Tools* manual describes the components found in the MS OS/2 Programmer's Toolkit and shows how you can use them to create MS OS/2 applications.

- Chapter 1, "Introduction," provides an overview of the programs and tools found in the MS OS/2 Programmer's Toolkit, and tells you where to find additional information about programming for MS OS/2.
- Chapter 2, "Building MS OS/2 Programs and Dynamic-Link Libraries," describes the steps involved in building an MS OS/2 application or dynamic-link library. It also provides programming examples.
- Chapter 3, "Miscellaneous Development Tools," describes the following programs, which aid in MS OS/2 program development: **kbdp**, **libwhere**, **mkmsgf**, **msgbind**, **shd** and **where**.
- Appendix A, "Using the QuickHelp Program," describes the QuickHelp program, which displays help information about MS OS/2 functions and topics.

1.5 Notational Conventions

This manual uses the following conventions for command arguments:

Convention	Use
<i>italics</i>	You must supply the text for any of the variables shown in italics. For example, when <i>option</i> appears in a syntax line, you should type the name of the option. Filenames and directory names also appear in italics.
bold	Bold type is used for names of programs, fields, data types, structures, statements, options, registers, and keywords. These items appear exactly as they would in code.
monospace	Items in monospace represent code excerpts or sample command lines.
[[brackets]]	Items in brackets are optional. To include optional information, type only the information within the brackets. Do not type the brackets themselves.
... (ellipsis)	An ellipsis (...) means that you can repeat an item as many times as necessary.

Chapter 2

Building MS OS/2 Programs and Dynamic-Link Libraries

2.1	Introduction	9
2.2	Setting Up Your Development Environment	9
2.3	Building an MS OS/2 Program	9
2.3.1	Editing Source Code	10
2.3.2	Compiling Source Code	11
2.3.3	Linking MS OS/2 Programs	11
2.3.4	Debugging MS OS/2 Programs	12
2.3.5	Binding Applications	13
2.3.6	Building Larger-Memory-Model Programs	13
2.3.7	The Make Program	14
2.4	Building a Dynamic-Link Library	15
2.4.1	Editing Source Code	16
2.4.2	Compiling Dynamic-Link Libraries	17
2.4.3	Creating Module-Definition Files	17
2.4.4	Supplying Export and Import Definitions	18
2.4.4.1	Using the Module-Definition File	18
2.4.4.2	Creating Import Libraries	18
2.4.5	Linking Dynamic-Link Libraries	19
2.4.6	Debugging a Dynamic-Link Library	20
2.5	Writing an MS OS/2 Assembly-Language Program	20
2.6	Writing an Assembly-Language Dynamic-Link Library	22
2.6.1	Setting Up a Stack Frame	23
2.6.1.1	Setting Up the Frame Pointer	24

2.6.1.2	Setting Up Space for Local Variables	24
2.6.1.3	Setting Up the Data Segment of a Dynamic-Link Library	24
2.6.1.4	Saving the SI, DI, and SS Registers	25
2.6.1.5	Pushing Parameters on the Stack	25
2.6.2	Restoring the DS and BP Registers upon Exiting	26
2.6.3	Adding an Initialization Function	26

2.1 Introduction

This chapter explains how to build programs and dynamic-link libraries for Microsoft Operating System/2. The first part of the chapter explains how to use the tools in the MS OS/2 Programmer's Toolkit to create an MS OS/2 program, and the second part contains programming examples that use these tools. In addition, this chapter explains how to use **make** description files to simplify program development, the special requirements for creating a dynamic-link library, and the changes needed for developing MS OS/2 applications in assembly language.

The programming descriptions and examples in this chapter are designed for programmers who are developing MS OS/2 C-language programs and using the Microsoft C Optimizing Compiler, version 5.1 or higher. You can also use other C compilers designed for MS OS/2, or you can develop programs in Microsoft BASIC, Pascal, or FORTRAN. For information about developing MS OS/2 applications written in these languages, see the documentation accompanying your language product.

In addition to the simple examples in this chapter, several sample programs are provided on the MS OS/2 Programmer's Toolkit distribution disks. These programs illustrate the use of MS OS/2 functions and programming techniques.

2.2 Setting Up Your Development Environment

To set up your development environment, you must first install the MS OS/2 Programmer's Toolkit software on your system. Use the instructions found on the "Welcome" page at the front of this binder.

2.3 Building an MS OS/2 Program

To create an MS OS/2 program, follow these basic steps:

1. Edit your source code.
2. Compile your program, using the Microsoft C Optimizing Compiler, version 5.1 or higher.
3. Link the program with the necessary run-time and import libraries.

4. Debug the program by using the Microsoft CodeView debugger.
5. Bind the program, if you want it to run in both real and protected modes.

Each of these steps is described in the following sections. In some cases, you may want to create an optional module-definition file for an MS OS/2 program. This type of file is described in the *Microsoft CodeView and Utilities Update* and in Section 2.4.3, "Creating Module-Definition Files," later in this chapter.

2.3.1 Editing Source Code

Edit your source code by using a text editor, such as the Microsoft Editor. This editor is supplied with every Microsoft language product that is available for use with MS OS/2.

The MS OS/2 Programmer's Toolkit provides an on-line help program, QuickHelp, which displays help information about MS OS/2 functions. You can display the syntax, description, and parameters for any MS OS/2 function from the command line or from a text editor. This help program makes it easy to get help information quickly while you are writing source code. The QuickHelp program is described in Appendix A, "Using the QuickHelp Program."

To illustrate how an MS OS/2 program is written, here is a source file *test.c* for a simple small-model program called **test**:

```
#include <os2.h>

main( ) {
    USHORT usBytes;
    DosWrite (1, "Hello\r\n", 7, &usBytes);
}
```

You must include the *os2.h* file in the program. This file, which contains definitions for MS OS/2 functions and structures, is included with the software for the MS OS/2 Programmer's Toolkit. The **USHORT** data type is defined in *os2def.h*. Make sure that you spell MS OS/2 function and structure names exactly as given in this file or in the *Microsoft Operating System/2 Programmer's Reference*; that is, you must preserve uppercase and lowercase letters as shown in these sources. Use casts where recommended, so that the compiler can convert the expression to the correct data type.

2.3.2 Compiling Source Code

You compile your C source code by using the **cl** command. For example, to compile and link the *test.c* file, type the following:

```
cl test.c
```

By default, if you type **cl filename**, your file will be both compiled and linked.

For protected-mode programs that run with MS OS/2, you might want to specify the **/G2** option, which enables 80286 instructions.

Some programs (such as **test**) can be compiled and linked from the **cl** command line. But for other programs, you should specify the **/c** (compile without linking) option on the **cl** command line so that your program is compiled without linking. For example, if your program requires a module-definition file, you must use the **/c** option to create an object file and then use that object file in a separate **link** command line.

To learn about compiler usage and options, see the documentation that accompanies your compiler. If you are compiling a dynamic-link library, see Section 2.4, “Building a Dynamic-Link Library.”

2.3.3 Linking MS OS/2 Programs

The Microsoft linker **link** can produce programs that run with DOS 3.x or MS OS/2. If you do not specify a module-definition file or import library as input, the linker produces a program for either DOS 3.x or MS OS/2, depending upon which libraries your program is linked with. If you specify a module-definition file that contains a **LIBRARY** statement, the linker produces a dynamic-link library. Otherwise, it produces a program for MS OS/2.

As mentioned previously, you can either link automatically on the **cl** command line or separately by invoking the **link** program. For example, the following command line compiles and links the *test.c* file:

```
cl test.c
```

The following command line, however, only compiles the file:

```
cl /c test.c
```

The compiler links this file with the default library that you built during the setup operation (or that you built with the **lib** program).

MS OS/2 programs must be linked with the *doscalls.lib* library. The Microsoft C Compiler automatically searches for and links with this library unless you specify the **/NOD** (no default library search) option on the **cl** command line. If this option is used, you must explicitly specify *doscalls.lib* and any other run-time libraries to link with.

The linker uses default filenames if none are specified in the link command line. For example, in the following **link** command line, the object file *test.obj* is linked with the default libraries *doscalls.lib* and *slibcep.lib* to produce the executable file *test.exe* and the map file *test.map*; the file *test.def* contains module-definition information:

```
link test,,,test.def;
```

For more information about the **link** program, see the *Microsoft CodeView and Utilities Update*, which is supplied with Microsoft language products.

2.3.4 Debugging MS OS/2 Programs

The Microsoft CodeView debugger has many features that are especially useful for debugging MS OS/2 applications, including the following:

- Structure watching
- Graphic Display command
- Dynamic-link module and multiple-thread debugging facilities

To prepare files for use with the CodeView debugger, you must specify the **/Zi** option on the **cl** command line and the **/CO** option on the **link** command line. These flags cause the compiler and linker to prepare the proper symbol and line-number information for CodeView.

To debug the **test** program by using CodeView, type the following:

```
cvp test
```

CodeView lets you view the source file, set breakpoints by line number, and view variables and structures by name. To learn about CodeView debugging commands and options, see the Microsoft CodeView and Utilities manual supplied with your Microsoft language product, and use the on-line help facilities.

2.3.5 Binding Applications

The MS OS/2 Programmer's Toolkit provides a program called **bind**, which lets you build applications that run in both real and protected modes. This program converts a protected-mode program into one that runs in both protected mode and real mode. However, only programs that make calls to functions in the Family API (application programming interface) can be converted. **Bind** replaces Family-API functions with simulator routines from the standard (object-code) library *api.lib*.

For more information about the **bind** program's syntax and options, see the *Microsoft CodeView and Utilities Update* supplied with your compiler.

2.3.6 Building Larger-Memory-Model Programs

The simplest type of MS OS/2 program to build is one that uses the small-memory model and calls only MS OS/2 functions. In the previous sections, you have seen how to build the small-model program **test**.

In addition to small-model programs, you can also set up medium-, compact-, large-, or huge-model programs. For example, if your program uses one data segment but more than one code segment, you must use the medium-memory model. This memory model is used for large programs whose code cannot fit into a single 64K segment, or for programs that do not require all of their code to be in memory at all times.

The following example shows how to build the medium-model program **medium**. It is composed of the files *medium1.c* and *medium2.c*. The *medium1.c* file contains the following:

```
#include <os2.h>

main( ) {
    Sample ("Hello\r\n", 7);
}
```

The *medium2.c* file contains the following:

```
#include <os2.h>

Sample (pchString, cbLength)
PCH pchString;
USHORT cbLength;

{
    USHORT usBytes;
    DosWrite(1, pchString, cbLength, &usBytes);
}
```

In this example, *medium1.c* calls the function **Sample**, which is located in *medium2.c*. The parameters *pchString* and *cbLength* are passed to the routine.

As with all MS OS/2 programs, the *os2.h* include file is included in the source files. Each source file must be compiled separately. The resulting code is then placed in separate segments by the linker. To direct the linker to do this, you must name the segments in a module-definition file. The module-definition file *medium.def* looks like this:

```
NAME medium1
SEGMENTS medium1_text medium2_text
```

The **SEGMENTS** statement gives the names of the code segments for *medium1.c* and *medium2.c*. These are the names created for both segments by the compiler.

You must compile the two source files as follows:

```
cl /c /AM medium1.c
cl /c /AM medium2.c
```

The **/c** option tells the compiler to compile without linking. The **/AM** option specifies the medium-memory model.

Now link the two object files with the proper libraries and the module-definition file, as follows:

```
link medium1.obj medium2.obj,,,doscalls.lib mlibcep.lib,medium.def
```

In this case, the default libraries *mlibcep.lib* and *doscalls.lib* are used. The *mlibcep.lib* library is a combined library for medium-model programs. It is analogous to the *slibcep.lib* library for small-model programs.

2.3.7 The Make Program

The **make** program helps to automate program development. This program, which is supplied with Microsoft compiler products, builds or updates an executable file for you from information that you supply in a **make** description file.

For the sample file *test.c*, your **make** description file would be very simple:

```
test.obj:test.c
      cl test.c
```

Make description files are more useful when you are compiling larger model programs, or programs that must be compiled and linked in separate steps. The **make** description file for the medium-model program **medium** looks like this:

```
medium1.obj:      medium1.c
                 cl /c /AM medium1.c

medium2.obj:      medium2.c
                 cl /c /AM medium2.c

medium1.exe:      medium1.obj medium2.obj
                 link medium1.obj medium2.obj, , doscalls.lib mlibcep.lib, medium.def
```

To build **medium**, you simply type the following:

```
make medium
```

This causes *medium1.c* and *medium2.c* to be compiled separately, then linked with the *doscalls.lib* and *mlibcep.lib* libraries and the module-definition file *medium.def*.

For details about how to create a **make** description file, see the Microsoft CodeView and Utilities manual supplied with your Microsoft compiler.

2.4 Building a Dynamic-Link Library

This section describes how to build a dynamic-link library. This type of library contains executable code for common functions, just as an ordinary library does. But code for dynamic-link functions is not linked into the executable file. Instead, the library itself is loaded into memory at run time, along with the executable file. Using a dynamic-link library saves memory, since the library is loaded only at run time, and more than one program may share the code.

To build a dynamic-link library, follow these steps:

1. Edit your source code.
2. Compile your dynamic-link library.
3. Create a module-definition file.
4. Create an import library (or use import definitions).
5. Link your dynamic-link library.
6. Debug the library by using the CodeView debugger.

These steps are discussed in the following sections. If you are writing a dynamic-link library in assembly language, see Section 2.6, “Writing an Assembly-Language Dynamic-Link Library.”

After creating the dynamic-link library, remember to copy it to a directory specified by the **libpath** command in your *config.sys* file.

2.4.1 Editing Source Code

To illustrate how a dynamic-link library is built, here is a simple dynamic-link library source file named *dynlink.c*:

```
#include <os2.h>
int _acrtused = 0;

far Sample (pchString, cbLength)
PCH pchString;
USHORT cbLength;

{
    USHORT usBytes;
    DosWrite (1, pchString, cbLength, &usBytes);
}
```

As with all MS OS/2 programs, the *os2.h* file is included in the library, since the MS OS/2 function **DosWrite** is called. Notice that the **_acrtused** variable is declared and initialized to zero. This directs the linker not to load the *crt0* startup module, which is not needed with dynamic-link libraries.

The library routine **Sample** is declared as a far segment. This is because the dynamic-link library resides in a different segment from that of the calling program.

Dynamic-link libraries can contain either global or local variables. If your library uses global variables, the data segment of the calling program must be pushed on the stack and then reloaded with the library's own data segment. You can do this by using the **/Asnu** options on the **cl** command line, or by using the **_loadds** keyword before a function name (both are discussed in the following section). Strings are always treated as global variables.

In addition to the standard C libraries, the Microsoft C Compiler version 5.1 offers special run-time libraries for use with single- or multiple-thread dynamic-link libraries. These libraries are described in a “readme” file supplied with the compiler.

2.4.2 Compiling Dynamic-Link Libraries

When compiling dynamic-link libraries, you must use the **/Asnu** options on the **cl** command line. These options direct the compiler to set near (16-bit) data pointers and addresses for all code and data items, and to set the **SS** register not equal to the **DS** register. This causes the current address in the **DS** register to be saved upon entry to a dynamic-link library function and restored upon exit.

Another way to set the stack and data segments for a dynamic-link library is to use the **_loadds** keyword before a function, as in the following example:

```
far _loadds Sample
```

By default, the **DGROUP** segment is loaded.

You must use the **/Gs** (remove stack probes) option when compiling a dynamic-link library. Because the stack setup for dynamic-link libraries differs from the stack setup for MS OS/2 programs, stack checking would cause unpredictable results. You must also use the **/c** option when compiling a dynamic-link library. This option compiles a file without linking it.

To compile the source file for the sample library *dynlink.c*, you would type the following command line:

```
cl /c /Asnu /Gs /Zl /Zi dynlink.c
```

The **/Zl** option prevents the default library names from being placed in the object file, and the **/Zi** option creates an object file that can be used with the CodeView debugger.

2.4.3 Creating Module-Definition Files

A module-definition file describes the name, attributes, exports, imports, and other characteristics of a program or library for MS OS/2. This file is optional for MS OS/2 programs, but is required for dynamic-link libraries.

Since you specify the module-definition file on the **link** command line, you must compile your source code with the **/c** option and link the object file separately.

For a complete listing and description of module-definition statements, see the *Microsoft CodeView and Utilities Update*.

2.4.4 Supplying Export and Import Definitions

Each dynamic-link library must use export definitions to make its functions directly available to other modules. Each executable file that calls a dynamic-link library must use import definitions that tell where each dynamic-link function can be found.

Export definitions can be supplied in two ways: in a module-definition file or in an import library. Import definitions are supplied in the calling program's module-definition file or in an import library.

2.4.4.1 Using the Module-Definition File

You can supply an export definition for the dynamic-link library in its module-definition file. For example, the dynamic-link library *dynlink.dll* contains the function **Sample**, which can be called from other programs. To make **Sample** available to calling programs, the module-definition file *dynlink.def* is as follows:

```
LIBRARY dynlink
EXPORTS _Sample
```

The **LIBRARY** statement tells the linker that the file *dynlink* is a dynamic-link library. The **EXPORTS** statement tells the linker that the function **Sample** in *dynlink* is available for other programs to call (the C compiler prepends the underscore to the function name). A program that calls the dynamic-link library *dynlink* must specify an import definition for **Sample** by using the following **IMPORTS** statement (unless an import library for it exists) in its module-definition file:

```
IMPORTS dynlink._Sample
```

You must list every function that will be called at run time in the module-definition file of the dynamic-link library.

2.4.4.2 Creating Import Libraries

An alternative way to specify import definitions is to create an import library. If you have many functions in a dynamic-link library, it is usually easier to create an import library than to use import definitions. The import library *doscalls.lib* is an example of an import library that is linked with all MS OS/2 applications.

To generate an import library, first create a module-definition file for the dynamic-link library. As in the previous example, it should contain **LIBRARY** and **EXPORTS** statements. Once you have created the module-definition file, use the **implib** program to create an import library to use during linking.

For example, to create the import library *dynlink.lib* for the dynamic-link library *dynlink*, run **implib** as follows:

```
implib dynlink.lib dynlink.def
```

If your program makes calls to the newly created dynamic-link library, its import library name must be included in the library field of the **link** command. For example, if the program **dcall** calls *dynlink.dll*, link **dcall** with the library as follows:

```
link /NOI dcall.obj,,,doscalls.lib dynlink.lib,dcall.def
```

2.4.5 Linking Dynamic-Link Libraries

To link dynamic-link libraries, follow these steps:

1. Specify the name of the dynamic-link library (*name.dll*) in the executable-file field of the **link** command.
2. Add a **LIBRARY** statement to the *.def* file to specify the dynamic-link library.
3. Add an **EXPORTS** statement to the *.def* file for the dynamic-link library.
4. Run the **implib** program to generate an import library for the dynamic-link library, or use an **IMPORTS** statement in the *.def* file of the calling program.
5. Use the **/NOI** option on the **link** command line if your dynamic-link library contains C functions.

You might also want to use the **/CO** option on the **link** command line if you plan to use the CodeView debugger.

To link the object file for the *dynlink* library, type the following:

```
link /NOI /CO dynlink.obj,dynlink.dll,,,doscalls slibcep,dynlink.def
```

The **/NOI** option tells the linker to distinguish between uppercase and lowercase letters. This means that the function **Sample** would be considered different from **sample**. The **/CO** option prepares the library for debugging. On the **link** command line, the name of the dynamic-link library (*dynlink.dll*) is used in the executable-file field. The library is linked with the *doscalls* and *slibcep* libraries, and the module-definition file *dynlink.def*.

In addition to the standard C libraries, the Microsoft C Compiler version 5.1 also supplies special run-time libraries for use with single- or multiple-thread dynamic-link libraries. These libraries are described in a “readme” file supplied with the compiler.

2.4.6 Debugging a Dynamic-Link Library

To prepare files for use with the CodeView debugger, you must specify the **/Zi** option on the **cl** command line and the **/CO** option on the **link** command line. These options cause the compiler and linker to prepare the proper symbol and line-number information for CodeView.

A dynamic-link library must be debugged in the context of a program that calls it. For example, suppose that the program **dcall** calls the *dynlink* library. To debug the library, type the following:

```
cvp /L dynlink dcall
```

If your program is written in C, the C source file is first displayed on the screen. To enter and view the dynamic-link library code, you should use the F8 (trace) key. While you are in the library code, you can change the **View** option to **Mixed** to view the library functions in assembly language.

For more information about CodeView, see the Microsoft CodeView and Utilities manual and the *Microsoft CodeView and Utilities Update*.

2.5 Writing an MS OS/2 Assembly-Language Program

You may choose to write MS OS/2 programs in assembly language instead of in the C language. This section describes the programming tools available for developing MS OS/2 programs in assembly language.

To aid in MS OS/2 program development, the Microsoft Macro Assembler, version 5.0 or higher, provides many useful directives. First, you can automatically link your program with the *doscalls.lib* library by specifying the following in your source code:

```
INCLUDELIB doscalls.lib
```

You can also enable the 80286 instruction set by using the **.286** directive at the top of your code.

The macro assembler provides simplified model and segment directives, as well. Use the **.MODEL** directive at the beginning of your source file, followed by a model type, to specify the model you are using. This directive automatically generates near or far addresses, depending upon the model selected. For example, the directive **.MODEL small** automatically generates near (16-bit) code and data addresses. The **.CODE**, **.DATA**, and **.STACK** directives can then be used to declare the code, data, and stack segments. Finally, you must make sure to declare all MS OS/2 functions to be **EXTERN:far**.

To show how an MS OS/2 program might be written in assembly language using some of these directives, the following program, *beep.asm*, writes the string "Hello" to the screen, causes the program to beep, and then exits:

```
.286
INCLUDELIB doscalls.lib
.MODEL small

Frequency      equ      660
Duration        equ      1000
ActionCode      equ      1
ExitCode        equ      1

.STACK 800h

.DATA
string db      "Hello", 13, 10
bytes  dw      0

.CODE
extrn  DosWrite:far
extrn  DosBeep:far
extrn  DosExit:far

start:
        push    1
        push    ds
        push    offset string
        push    7
        push    ds
        push    offset bytes
        call    DosWrite
        push    Frequency
        push    Duration
        call    DosBeep
        push    ActionCode
        push    ExitCode
        call    DosExit

        END     start
```

The parameters for the **DosWrite**, **DosBeep**, and **DosExit** functions are pushed on the stack before the calls to the functions are made. The Pascal calling convention is required for MS OS/2 functions. This means that the parameters are pushed in the order in which they appear in a function. For example, the parameters for **DosBeep** (*Frequency*, *Duration*) are pushed on the stack as follows:

```
push    Frequency
push    Duration
call    DosBeep
```

To assemble and link this program by using default options, use the following commands:

```
masm beep;
link beep;
```

If you do not specify a filename extension to the assembler, the extension *.asm* is used by default. The semicolon at the end of the command line causes the assembler and linker fields to be filled with default options.

To aid in MS OS/2 program development, the Microsoft Macro Assembler provides files that define structures, constants, and macros. These files let you make calls to MS OS/2 functions in a simpler way. For more information about how to use include files, see the documentation supplied with your assembler.

2.6 Writing an Assembly-Language Dynamic-Link Library

You can also use assembly language to write a dynamic-link library. If you choose to do so, you must declare the functions in your library as follows:

function **proc far**

You should declare the functions to be **public**.

The rest of this section explains how to set up a stack frame for your dynamic-link library, how to restore registers upon exiting an MS OS/2 function, and how to add an initialization function to your dynamic-link library.

2.6.1 Setting Up a Stack Frame

In addition to declaring functions, your assembly-language dynamic-link library must set up a stack frame for the arguments being passed to it. The elements in a stack frame are shown in Figure 2.1:

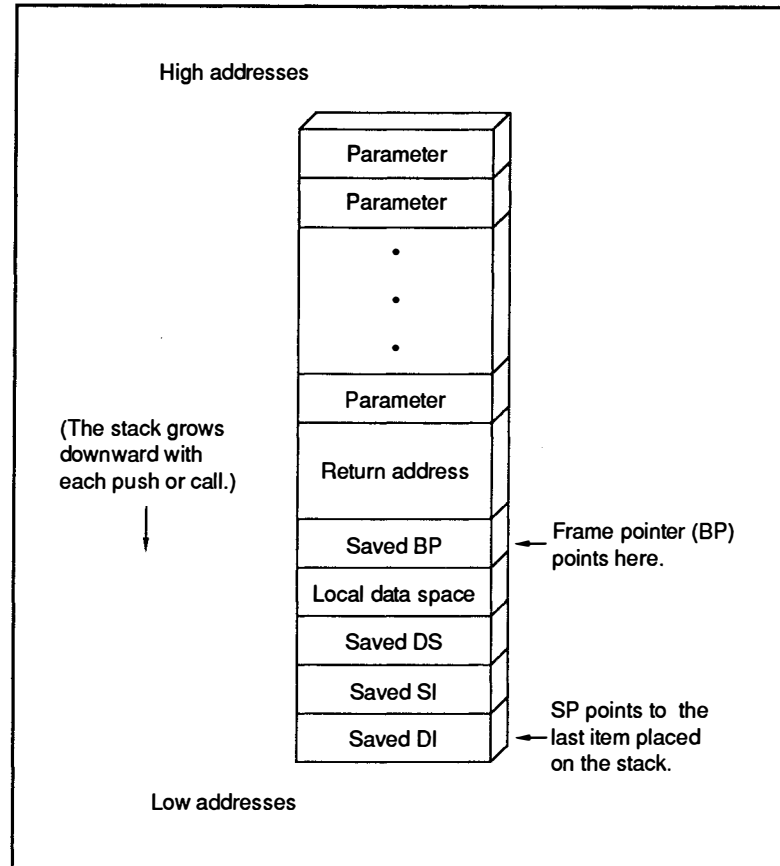


Figure 2.1 The Stack Frame

To set up the stack frame, the code in your dynamic-link library must do the following:

- Set up the frame pointer (**BP**).
- Set up space for local variables.
- Save the old **DS** register and set up the data segment for your dynamic-link library.

- Save the **SI**, **DI**, and **SS** registers (if the values will be changed).
- Push MS OS/2 parameters on the stack.

The following sections describe how to set up a stack frame and call MS OS/2 functions in assembly-language dynamic-link libraries. For more information about setting up a stack frame for assembly language procedures, see the *Microsoft Mixed-Language Programming Guide*, which is supplied with your compiler.

2.6.1.1 Setting Up the Frame Pointer

Upon entry to the dynamic-link library, functions must first set up a frame pointer by pushing the old **BP** register value on the stack and setting the **BP** register equal to the **SP** register, as follows:

```
push    bp
mov     bp, sp
```

2.6.1.2 Setting Up Space for Local Variables

If your called function needs local variables, they are also pushed on the stack. After pushing the old **DS** register value on the stack and setting the **BP** register equal to the **SP** register, move the stack pointer down and push local variables on the stack. For example, the following lines save the old **BP** value, assign the **BP** register to **SP**, and move **SP** so that it points to local variables:

```
push    bp
mov     bp, sp
sub     sp, 04
```

Local variables are pushed on the stack in the positions `[bp-2]`, `[bp-4]`, `[bp-6]`, and so on. Thus, you use these addresses to access local variables.

Upon exiting, the function should restore **SP** by setting it equal to **BP**.

2.6.1.3 Setting Up the Data Segment of a Dynamic-Link Library

If you are creating a dynamic-link library that has an automatic data segment, you must set the **DS** register to the proper value when the function starts. To do this, set **DGROUP** so that it points to the data segment of the dynamic-link library, as follows:

```
DGROUP  GROUP    data
```

Each function in the dynamic-link library that needs to use the library's automatic data segment must save the old data segment and reload the **DS** register so that it points to the data segment of the dynamic-link library. Placed at the beginning of the function, the following lines of code do this:

```
push    ds
mov     ax, seg DGROUP
mov     ds, ax
```

Before the function returns to the calling program, it must restore the old **DS** register value by popping it off the stack, as follows:

```
pop     ds
```

2.6.1.4 Saving the SI, DI, and SS Registers

Upon entry to the dynamic-link library, functions should save the **SI**, **DI**, and **SS** registers by pushing them on the stack, after setting the frame pointer and allocating any existing local data. (It is not necessary to push these registers on the stack if the functions in your library do not change them.) The following example shows how to push them on the stack:

```
push    bp
mov     bp, sp
sub     sp, 4
push    si
push    di
push    ss
```

Upon exiting the dynamic-link library, functions must pop the registers off the stack in reverse order:

```
pop     ss
pop     di
pop     si
```

2.6.1.5 Pushing Parameters on the Stack

Parameters for MS OS/2 functions in a dynamic-link library are pushed on the stack before the call is made. Since MS OS/2 functions require that you use the Pascal calling convention, parameters are pushed on the stack in the order in which they appear in the function.

2.6.2 Restoring the DS and BP Registers upon Exiting

When the calling program exits from a function for which parameters were pushed on the stack, the called function should always restore the **DS** and **BP** registers, then use a **RET** *nn* instruction, where *nn* is the number of bytes pushed upon exit.

Return values are returned in the **AX** (16-bit) register or in the **DX:AX** (32-bit) registers, where **DX** holds the high 16 bits and **AX** holds the low 16 bits.

2.6.3 Adding an Initialization Function

You can optionally add an initialization function to a dynamic-link library. This function, which is called before the actual code for the dynamic-link library is called, performs user-defined operations. As with dynamic-link library functions, you must push existing registers on the stack before using them in an initialization routine, and you must set automatic data segments as described in Section 2.4.2, "Compiling Dynamic-Link Libraries."

Be aware that when MS OS/2 starts executing a dynamic-link library, it sets the CPU registers to known values. For a list of these initial values, see the *Microsoft Operating System/2 Programmer's Reference*.

Initialization functions must be written in assembly language. This is because the assembly-language **end** statement signals the start of the library. In C, there is no equivalent end statement. You can, however, write your initialization code in assembly language and the library functions in C. Assemble and compile these files separately, then combine them during linking.

Upon exiting, your program should restore the stack in reverse order.

Chapter 3

Miscellaneous Development Tools

3.1	Introduction	29
3.2	The Kbdp Program	30
3.3	The Libwhere Program	31
3.4	The Mkmsgf Program	32
3.5	The Msgbind Program	34
3.6	The Shd Program	36
3.7	The Where Program	39

—

—

—

3.1 Introduction

Microsoft Operating System/2 provides several tools that will help you develop your own programs. This chapter describes the following MS OS/2 programs:

- **kbdp**
- **libwhere**
- **mkmsgf**
- **msgbind**
- **shd**
- **where**

Each of these MS OS/2 programs is described according to the following information, although some of the programs are more complex and may include additional information (see **mkmsgf** and **msgbind**):

- Syntax
- Purpose
- Parameters
- Examples

Another program, QuickHelp, displays help information for MS OS/2 functions. For information about QuickHelp, see Appendix A, “Using the QuickHelp Program.”

3.2 The Kbdp Program

■ **kbdp** *delay rate*

The **kbdp** program changes the typematic rate of the keyboard. If you type **kbdp** without parameters, the current settings are displayed.

Parameter	Description
<i>delay</i>	Specifies the delay in milliseconds (1–1000) before repeating keystrokes are displayed.
<i>rate</i>	Specifies the repeat rate per second (2–30 characters/sec).

3.3 The Libwhere Program

■ libwhere [/c] *library*

The **libwhere** program searches the configuration command **libpath** for filenames that match the specification on the command line. The **libwhere** program locates an MS OS/2 library by looking in the appropriate *config.sys* file for the path that follows the **libpath=** string, then creates an environment variable with this path and calls the **where** program to search it.

The **libwhere** program sends messages to the standard error file, and sends the list of filenames to the standard output file. This means that you can redirect the list of filenames to other programs that take a list of filenames from the standard input file (for example, the **exehdr** program).

Parameter	Description
/c	Specifies which <i>config.sys</i> file to search. You might want to use this option if you run libwhere in real mode.
<i>library</i>	Specifies the library to search for. The <i>library</i> parameter can be any valid MS OS/2 library name.

Example

The following protected-mode command finds all filenames that have a *.dll* extension. The only directories it searches are those listed in the **libpath** setting of the *config.sys* file in the root directory of the boot device:

```
libwhere *.dll
```

The following command finds the **.dll* files as in the previous example and pipes them to the **exehdr** program for analysis:

```
libwhere *.dll | exehdr -
```

In either protected or real mode, the following command also finds all filenames that have a *.dll* extension. The only directories it searches are those listed in the **libpath** setting, in the *config.sys* file in the root directory of drive C:

```
libwhere /c c:\config.sys *.dll
```

3.4 The Mkmsgf Program

■ **mkmsgf** *infile outfile*

The **mkmsgf** program reads the input message file *infile* that you specify, and creates an output message file *outfile* that the **DosGetMessage** function can use to display messages.

You can use the output message file in two ways to display messages. First, you can specify a message filename and a message number in the **DosGetMessage** parameter list. Or you can bind the output message file to the message segment of an executable file. The description of the **msgbind** program, later in this chapter, tells you how to do this.

Parameter	Description
<i>infile</i>	Specifies the input file that contains message profiles. The input filename can be any valid MS OS/2 filename, optionally preceded by a drive letter and a pathname. The <i>infile</i> file should have a .msg extension. The format of an input file is described in the following section, "Message File Format."
<i>outfile</i>	Specifies the output file created by mkmsgf . This file contains the indexed message file that DosGetMessage will use. The output filename can be any valid MS OS/2 filename, optionally preceded by a drive letter and a pathname. The output file cannot have the same name as the input file.

Message File Format

The input message file *infile* is a standard ASCII file that contains the following three types of lines:

- Comment
- Component ID
- Component message

Comment lines are allowed only at the beginning of the file, before the line that contains the component ID. Comment lines must begin with a semi-colon (;) in the first column of the line.

A component ID line contains a three-character name identifier (for example, DOS), which precedes all **mkmsgf** message numbers.

Component message lines consist of a message header and an ASCII text message. The message header is the component ID, followed by a four-digit message number, a single letter specifying the type of help message (E, H, I, P, or W), a colon (:), and a blank space. For example, DOS0100E: is DOS error message 100. The message header must begin in the first column of the line. Only one header must precede the text of a message, although a message can span multiple lines.

The following is a list of message types:

Message type	Meaning
E	Error
H	Help
I	Information
P	Prompt
W	Warning

Message numbering may start at any number, but messages must be numbered sequentially. If you do not use a message number, you must insert an empty entry in its place in the text file.

If you insert the character sequence %0 at the end of a message line, a return/linefeed will not be executed after the message is displayed. This lets you prompt a user for input on the same line as the message.

Help-message filenames should begin with the component ID followed by *h.msg*. For example, the help file associated with the component identifier DOS would be *dosh.msg*.

Example

The following is an example of a message file:

```
; This is an example of a
; message file for component DOS
; starting with three comment lines.
DOS
DOS0100E: File not found
DOS0101E:
DOS0102H: Usage: del [drive:]pathname
DOS0103E:
DOS0104I: %1 files copied
DOS0105E:
DOS0106W: Warning! All data will be destroyed!
DOS0107E:
DOS0108E:
DOS0109E:
DOS0110P: Do you wish to apply these patches (Y or N)? %0
DOS0111E: Divide overflow
```

3.5 The Msgbind Program

■ **msgbind** *infile*

The **msgbind** program binds a message segment to an executable program. It does this by reading an input file that specifies which executable files to modify. For each executable file, **msgbind** specifies which message files to scan, and for each message file, it specifies which messages to include in the executable file.

Parameter	Description
<i>infile</i>	Specifies the input file that contains the executable files, output message files, and message numbers that will be bound. The <i>infile</i> parameter can be any valid MS OS/2 filename, including an optional filename extension. The input file format is described in the following section, "Input File Format."

Input File Format

The input file *infile* contains the following three types of lines:

- > Executable file
- < Message file
- Message numbers

Each line may be repeated one or more times, subject to the rules described in the following paragraphs. Thus, more than one executable file can be modified by the specifications found in one input file.

The executable file that will be modified is preceded by a greater-than sign (>). Until it encounters another greater-than sign, **msgbind** continues to use this file. The name of the file can be any valid MS OS/2 filename.

The message file to be read from is preceded by a less-than sign (<). You create this file by using the **mkmsgf** program. The name can be any valid MS OS/2 filename. All message numbers that follow it are located in the specified message file and are copied to the current output executable file. **Msgbind** reads the message number list until it encounters one of the following: the end of the input file, a new output specification, or a new input-message file.

The messages in the message file are listed below the message filename. Only those message numbers that you specify will be added. Message numbers should consist of a three-letter component ID followed by a four-digit message number.

Example

The following is a **msgbind** input file. The first line specifies that the executable file to modify is *cmd.exe*. The messages DOS0100, DOS0123, and DOS0245 are read from the file *dosutil.msg* and added to the *cmd.exe* file. The **msgbind** program then encounters an executable-file option for the *format.exe* file. The messages DOS0001 and DOS0006 are read from *dosutil.msg* and added to *format.exe*. Finally, the messages FMT0001 and FMT0002 are read from the file *format.msg* and added to *format.exe*.

```
>c:\cmd.exe
<c:\os20\dosutil.msg
DOS0100
DOS0123
DOS0245
>c:\format.exe
<c:\os20\dosutil.msg
DOS0001
DOS0006
<c:\format.msg
FMT0001
FMT0002
```

The files *dosutil.msg* and *format.msg* in this example are two output-message filenames from the **mkmsgf** program. For more information about creating message files, see the **mkmsgf** program, earlier in this chapter.

3.6 The Shd Program

■ **shd** [*-format*] [*-mode*] [*-s offset*] [*-n count*] *name* [...]

The **shd** program displays the contents of named shared memory segments or files. The contents can be displayed in hexadecimal, octal, decimal, or character format.

The default setting for the format and base options is **-abxA**. This means that addresses and bytes are printed in hexadecimal along with their ASCII character equivalents.

Format options may specify addresses, characters, bytes, words (two bytes), or long words (four bytes) to be printed in hexadecimal, decimal, or octal format. You can also specify two special formats: text or ASCII; and you may combine and repeat format and base options, as well. All format options that appear in a single argument apply as appropriate to all other options in that argument.

If you do not specify *name*, **shd** reads from the standard input file. If you specify one or more names, **shd** first checks to see if *name* is a valid shared memory segment. If a shared memory segment does not exist, **shd** checks for a file with that name. Once it finds a valid shared segment or file, **shd** treats all other names on the command line in the same manner as it did the first.

Parameter	Description														
<i>-format</i>	Specifies the format in which to display memory segments or files. Format options display output in the following formats:														
	<table><tr><th>Option</th><th>Meaning</th></tr><tr><td>a</td><td>Address</td></tr><tr><td>c</td><td>Character</td></tr><tr><td>b</td><td>Byte</td></tr><tr><td>w</td><td>Word</td></tr><tr><td>l</td><td>Long word</td></tr><tr><td>A</td><td>ASCII</td></tr></table>	Option	Meaning	a	Address	c	Character	b	Byte	w	Word	l	Long word	A	ASCII
Option	Meaning														
a	Address														
c	Character														
b	Byte														
w	Word														
l	Long word														
A	ASCII														

The base options, which determine the number base to use, are listed as follows:

Option	Meaning
x	Hexadecimal
d	Decimal
o	Octal
t	Text

If you select address (**-a**) output without other format options, **-bx** is assumed. If you select ASCII (**-A**) output, base options have no meaning. If you do not specify any base options, **-xdo** is used by default.

For text files, control characters with a value from 0x00 to 0x1F are printed as symbols from “^@” to “^_”. Bytes with the high bit set are preceded by a tilde (~) and printed as if the high bit were not set. The special characters (^, ~, and \) are preceded by a backslash (\) to preserve their special meanings. As special cases, two values are represented numerically by “\177” and “\377”. The **-t** option overrides all output format options except addresses.

-mode Overrides the default mode, which is to check first for a named shared segment and then for a file that matches the first *name* argument. The choices are **-m**, which means search only for a named shared memory segment, and **-f**, which means search only for a file. If it does not find a segment or file, **shd** displays an error message.

-s offset Specifies the beginning offset in the memory segment or file where printing is to begin. If you do not specify a memory segment or file argument, or if a search fails because the input is a pipe, **shd** reads the *offset* bytes from the input and discards them. Otherwise, a seek error terminates processing of the current memory segment or file.

You may specify the offset in decimal, hexadecimal (preceded by 0x), or octal (preceded by 0) format, optionally appended by one of the following multipliers:

Multiplier	Meaning
w	Word
l	Long word
b	Bytes (512-byte blocks)
K	1024 bytes

Any offset and multiplier may be optionally separated by an asterisk (*).

- n count** Specifies the number of bytes to process. The *count* parameter is in the same format as the *offset* parameter.
- name** Specifies the name of the file to display. You may specify more than one name. Names listed on the command line follow standard MS OS/2 file-naming conventions. If the name is a shared memory segment, do not include the "SHAREMEM\" part of the name, because **shd** automatically appends that part to the *name.ext* file.

Example

The following command displays the contents of the file *test.c* in hexadecimal byte-address format:

```
shd test.c
```

To display the same file in character format, type the following:

```
shd -c test.c
```

Instead of displaying hexadecimal numbers, the previous command displays their ASCII character equivalents.

The following command displays the address and bytes (in both hexadecimal and ASCII formats) for the file *test.c*, starting at the word offset 0x0020 for 0x0010 words. Since no format options are listed, the default setting **-abxA** is used.

```
shd -s 0x0020*w -n 0x0010*w test.c
```

3.7 The Where Program

■ **where** [/r *directory*] [/qte] *file...*

The **where** program searches for files and displays information about them. It can also display other useful information if you specify options on the command line.

By default, **where** attempts to find the specified files by searching the PATH environment variable. But if a pattern is of the following form, **where** searches the environment name to find the specified file:

\$environment-name:filename

Parameter	Description
/r	Searches recursively.
<i>directory</i>	Specifies which directory to search.
/q	Specifies whether to exit with 0 if the file is found, or to exit with 1 if it is not found.
/t	Displays the size of the file, and the date and time it was created or last modified.
/e	Displays the type of executable file.
<i>file</i>	Specifies the name of the file(s) to search for.

If you do not specify the /r option, **where** attempts to find files that match the patterns by searching the environment search strings.

If you do specify the /r option, **where** searches *directory* and its subdirectories looking for files that match the patterns. When you specify /r, a pattern may have only a filename part.

The filename part of the pattern may contain wildcards (* and ?).

Example

The following command finds all files that contain the letters **com** in the current path:

```
where com
```

You can also specify a different path. For example, the following command finds all files that contain the letters **com** in the path *tools*:

```
where $tools:com
```

The following command finds all occurrences of **com** on the current drive, starting from the root directory:

```
where /r \ com
```

To locate all occurrences of **com** by searching recursively through the current drive, type the following:

```
where /r . com
```

Or to locate all occurrences of **com** in drive D in the directory *\dos\tools*, type the following:

```
where /r d:\dos\tools com
```

Appendix A

Using the QuickHelp Program

A.1	Introduction	43
A.2	The MS OS/2 System Functions Database	43
A.3	Installing QuickHelp	44
A.4	Starting QuickHelp	45
A.5	Quitting QuickHelp	46
A.6	Getting Help with QuickHelp	46
A.7	Choosing Commands in QuickHelp Menus	46
A.8	Choosing a Category	47
A.9	Choosing a Topic from a Topic List	48
A.10	Viewing a Topic	49
A.11	Choosing a Topic from the QuickHelp Window	49
A.12	Choosing a Topic from the Screen	50
A.12.1	Setting the Window Height	50
A.12.2	Changing the Screen's Maximum Height	50
A.12.3	Hiding the QuickHelp Window	51
A.13	Searching for a Topic	51
A.14	Viewing Related Topics	52
A.14.1	Viewing the Next Topic	53
A.14.2	Returning to the Last Topic	54
A.14.3	Viewing a History of Your Session	54
A.15	Searching Within a Topic	54
A.16	Pasting from the Database	55
A.16.1	Setting the Paste-File Mode	56
A.16.2	Renaming the Paste File	56
A.17	Opening a New Database	56

A.18	Using QuickHelp with the Microsoft Editor	57
A.18.1	Installing QuickHelp in the Microsoft Editor	57
A.18.2	Opening the QuickHelp Window	57
A.18.3	Searching for a Topic from the Microsoft Editor	58
A.18.4	Pasting from the Microsoft Editor	58
A.19	Specifying Parameters on the Command Line	59
A.20	Using the QH Environment Variable	61

A.1 Introduction

QuickHelp is the display program for Microsoft on-line documentation databases. You use QuickHelp to search for and view information in a database. QuickHelp lets you do the following:

- View topics in a database by selecting the topic from a list of topics.
- Search for topics by entering or selecting a name either in the QuickHelp window or on your program screen.
- Copy examples and syntax from a topic to a paste file so that you may paste the section into your own source files.
- Display topics, search for topics, and paste examples and syntax in your source files while working in the Microsoft Editor.

QuickHelp reads Microsoft on-line documentation databases. A database is a collection of topics. A topic is one or more lines of text that describe a specific item, such as a function, command, or concept. Topics in a database are organized in categories. Typically, topics in a category are related in some way. With QuickHelp, you select the category you want to view, then select the topic or topics you want to view within that category. Selecting a category does not limit your access to the rest of the database. QuickHelp lets you move between categories quickly, and even lets you search all categories for specific topics at the same time.

A.2 The MS OS/2 System Functions Database

QuickHelp is the tool you need to view the MS OS/2 System Functions database in the MS OS/2 Programmer's Toolkit. The MS OS/2 System Functions database contains descriptions and definitions for all of the functions, structures, and data types in the MS OS/2 application programming interface. In particular, it contains the following categories:

Category	Content
MS OS/2 Overviews	Describes the MS OS/2 system functions.
Dos	Describes the MS OS/2 kernel functions.
Kbd	Describes the MS OS/2 keyboard functions.
Mou	Describes the MS OS/2 mouse functions.
Vio	Describes the MS OS/2 video input and output functions.

IOCtl	Describes the MS OS/2 input and output control functions.
Macros	Describes the utility macros provided with the MS OS/2 C-language header files.
Microsoft Editor	Describes the Microsoft Editor commands.
MS OS/2 Tools	Describes the MS OS/2 utilities provided with the MS OS/2 Programmer's Toolkit.

A.3 Installing QuickHelp

You install QuickHelp by copying the program and data files to directories on your hard disk, as shown by the following list:

File	Description
<i>qh.exe</i>	QuickHelp program file. Copy to a directory specified by your PATH variable.
<i>qh.dll</i>	QuickHelp dynamic-link library. Copy to a directory specified by your LIBPATH variable.
<i>mshelp.dll</i>	Microsoft Help library. Copy to a directory specified by your LIBPATH variable.
<i>qh.hlp</i>	MS OS/2 System Functions database. Copy to a directory specified by your PATH variable.

After installing QuickHelp, you can significantly enhance the speed with which QuickHelp loads and displays topics by taking the following steps:

- Use the **diskcache** statement in your *config.sys* file to create a disk cache. Although the cache can be any size, at least 256K is recommended.
- Make sure the directory containing the QuickHelp program and database files is the first directory in the PATH variable.
- If possible, place all QuickHelp files in the directory specified by the LIBPATH variable.

A.4 Starting QuickHelp

To start QuickHelp, type **qh** and press ENTER. QuickHelp loads the MS OS/2 System Functions database and opens the QuickHelp window. QuickHelp should look like this:

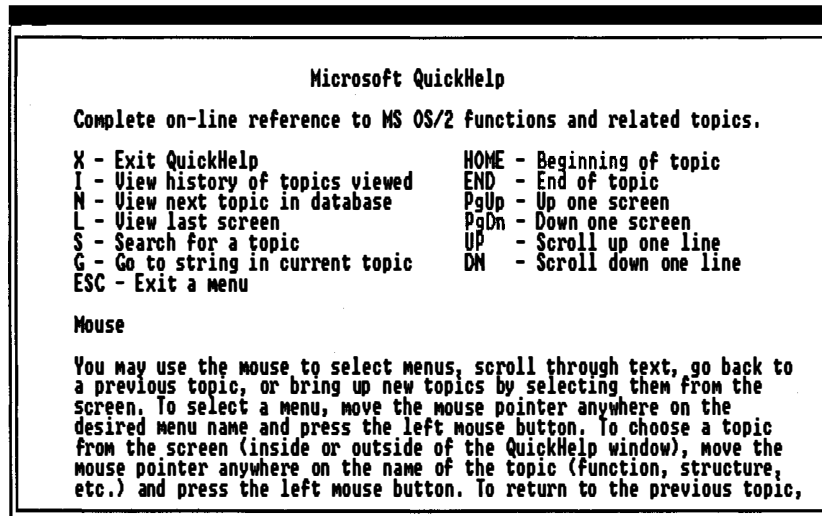


Figure A.1 QuickHelp Window

The first topic QuickHelp displays is the Microsoft QuickHelp help topic. This help topic explains how to use QuickHelp commands to view topics in the database.

You can also start QuickHelp from a text editor, such as the Microsoft Editor. The steps you use depend on the editor. For more information about starting QuickHelp from the Microsoft Editor, see Section A.18, "Using QuickHelp with the Microsoft Editor."

QuickHelp has several command-line options that you may use to change the default database, change the default paste filename, and set the QuickHelp window height. The QuickHelp command-line options are described in detail in Section A.19, "Specifying Parameters on the Command Line."

A.5 Quitting QuickHelp

You can quit QuickHelp by choosing the Exit QuickHelp command in the File menu. This command closes any open databases, restores the screen to its previous appearance, and exits QuickHelp. Once you exit QuickHelp, keyboard and mouse input to your previous program is restored.

QuickHelp does not save the window height, history, paste filename, or database name from session to session. This means that when you quit QuickHelp, these values are lost. To prevent having to reset these values each time you start QuickHelp, you can use the QuickHelp environment variable QH to define your preferred values as the QuickHelp default values. For more information on the QH environment variable, see Section A.20, "Using the QH Environment Variable."

Press X to choose the Exit QuickHelp command without selecting the File menu.

A.6 Getting Help with QuickHelp

You can display help information about QuickHelp by pressing the F1 key or by pressing H.

A.7 Choosing Commands in QuickHelp Menus

The QuickHelp menu bar, at the top of the window, lists the QuickHelp menus. The QuickHelp menus list the commands you need in order to choose categories and view topics. They also list commands that let you set the current window height, database, paste filename, and other program options.

To use a QuickHelp command, you must select the appropriate menu and choose the command from the list of commands in the menu. You can select menus and choose commands with either the keyboard or the mouse.

To choose a command with the keyboard, do the following:

1. Type the letter in the menu name that has the highlight or underscore. This selects the menu and directs QuickHelp to display the menu commands. QuickHelp places a highlight on the first command in the menu.

2. Use the UP or DOWN key to move the highlight to the command that you want.
3. Press the ENTER key to carry out the command, or the ESCAPE key to cancel the command.

To choose a command with the mouse, do the following:

1. Point to the menu name and click the mouse button. This selects the menu and displays the menu commands.
2. Point to the command and click the mouse button to carry out the command. To cancel the command, point to an empty part of the menu bar or window and click the mouse button.

You are free to combine the keyboard and mouse methods to choose commands. For example, you can select a menu with the keyboard and choose the command with the mouse, or vice versa.

After selecting a menu, you can select another, without first canceling the original menu, by using the LEFT and RIGHT keys. These keys move the selection left or right along the menu bar, removing the selection of the current menu and selecting the next menu.

A.8 Choosing a Category

You can choose a category for viewing by selecting the Categories menu and choosing a category from the list of categories in the menu. When you choose a category, QuickHelp displays that category's topic list. The topic list is an alphabetical list of names with accompanying descriptions for each topic in the category. You may then choose a topic for viewing by moving the highlight to the topic and pressing the ENTER key, or by pointing the mouse to the topic and clicking the mouse button. Figure A.2 shows the Kbd category selected in the Categories menu:

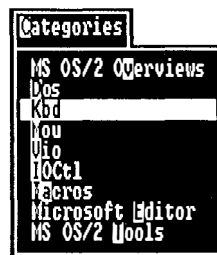
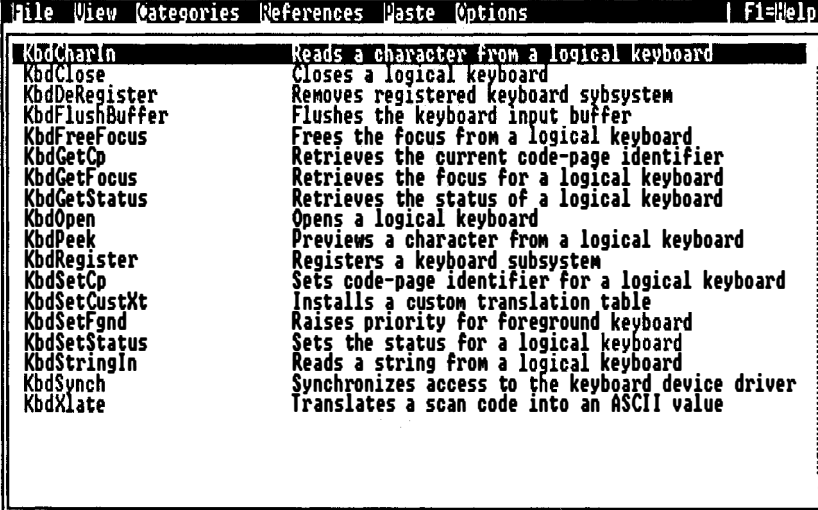


Figure A.2 Categories Menu

A.9 Choosing a Topic from a Topic List

You can choose a topic from a topic list by using the keyboard or the mouse. To choose a topic with the keyboard, use the UP or DOWN key to highlight the topic you want to view, then press the ENTER key. To choose a topic with the mouse, move the mouse pointer to any point on the same line as the topic and click the mouse button. In either case, QuickHelp loads the topic from the database and displays it in the QuickHelp window. Figure A.3 shows the Kbd topic list in the QuickHelp window:



Function	Description
KbdCharIn	Reads a character from a logical keyboard
KbdClose	Closes a logical keyboard
KbdDeRegister	Removes registered keyboard subsystem
KbdFlushBuffer	Flushes the keyboard input buffer
KbdFreeFocus	Frees the focus from a logical keyboard
KbdGetCp	Retrieves the current code-page identifier
KbdGetFocus	Retrieves the focus for a logical keyboard
KbdGetStatus	Retrieves the status of a logical keyboard
KbdOpen	Opens a logical keyboard
KbdPeek	Previews a character from a logical keyboard
KbdRegister	Registers a keyboard subsystem
KbdSetCp	Sets code-page identifier for a logical keyboard
KbdSetCustXt	Installs a custom translation table
KbdSetFgnd	Raises priority for foreground keyboard
KbdSetStatus	Sets the status for a logical keyboard
KbdStringIn	Reads a string from a logical keyboard
KbdSynch	Synchronizes access to the keyboard device driver
KbdXlate	Translates a scan code into an ASCII value

Figure A.3 Kbd Topic List

After QuickHelp displays a topic, you may see additional disk activity as it preloads topics related to the current topic. QuickHelp makes an educated guess about what you might want to see based on your current topic. To shorten the length of time between your selecting a topic and QuickHelp displaying it, QuickHelp loads these related topics while it is reading your keystrokes and mouse clicks.

A.10 Viewing a Topic

Once you choose a topic, QuickHelp displays the topic in the QuickHelp window. If the topic is too long to fit in the window, you can use the scrolling keys to scroll the topic one line at a time or one page at a time. A page is the same number of lines currently visible in the QuickHelp window. The following list describes the scrolling keys:

Key	Description
UP	Scrolls up one line. Use this key to move back to the beginning of the topic one line at a time.
DOWN	Scrolls down one line. Use this key to move forward to the end of the topic one line at a time.
PAGE UP	Moves up one full page. Use this key to move back to the beginning of the topic one full page at a time.
PAGE DOWN	Moves down one full page. Use this key to move forward to the end of the topic one full page at a time.
HOME	Moves to the first full page in the topic.
END	Moves to the last full page in the topic.

You can also scroll the topic by placing the mouse pointer on the scroll bar on the right side of the QuickHelp window and clicking above the scroll box (the gray box on the scroll bar) to scroll up, or below it to scroll down. If you press and hold down the mouse button while the pointer is on the scroll bar, QuickHelp continues to scroll the topic until it reaches the beginning or the end.

You may notice the scroll box on the scroll bar move while you scroll a topic. The box marks the position of the current full page of topic relative to the beginning of the topic. When the topic's last (or only) page is in view, the scroll box is at the bottom of the scroll bar. If the first full page is in view, the scroll box is at the top.

A.11 Choosing a Topic from the QuickHelp Window

You can also choose a topic for viewing by moving the mouse pointer to any word in the QuickHelp window and clicking the mouse button. QuickHelp reads the word from the window and searches the database for a topic with the same name. If QuickHelp finds the topic, it displays the topic. Otherwise, it displays the message "*topic* not found."

A.12 Choosing a Topic from the Screen

You can choose a topic for viewing from the screen by first setting the QuickHelp window height to less than the screen's maximum height, then using the mouse to choose a topic. Setting the window height to less than the maximum exposes a part of the screen that appeared immediately before you started QuickHelp. This means you can choose topics to search for from your work on the screen. You choose topics from the screen in the same way as you would from a window.

A.12.1 Setting the Window Height

You can set the height of the QuickHelp window by choosing the Window Height command in the Options menu. This command sets the height of the window to the number of lines you supply. The window height can be from five lines to the maximum number of lines supported by your screen. This command is useful if you want to choose topics for viewing from the screen.

To set the window height, follow these steps:

1. Choose the Window Height command.
QuickHelp displays a dialog box prompting you for the height of the window.
2. Enter the height of the window, in lines, and press the ENTER key.

If the height you supply is less than five but not zero, QuickHelp sets the window height to 5. If it is zero, or greater than the maximum height of the screen, QuickHelp sets the height to the screen height.

A.12.2 Changing the Screen's Maximum Height

You can change your screen's maximum height by choosing the Window Height command in the Options Menu and supplying a maximum screen height. Unlike setting the window height, setting the maximum screen height changes the display mode of your screen. QuickHelp restores the screen to the previous maximum height when it exits.

To set the maximum screen height, use a plus sign (+) at the beginning of the number you supply for the window height. QuickHelp interprets the number as the new maximum screen height. If the display driver for the screen does not permit the given number of lines, QuickHelp displays the message "Display doesn't support requested mode."

A.12.3 Hiding the QuickHelp Window

You can temporarily hide the QuickHelp window by choosing the Hide Window command in the View menu. This command hides the QuickHelp window and displays the entire screen that appeared immediately before you started QuickHelp. You may then use the keyboard or the mouse to choose topics for viewing from the screen.

Although the QuickHelp window is hidden, QuickHelp continues to receive all keyboard and mouse input. Therefore, if you were previously in an editing session, hiding the QuickHelp window would not let you continue editing the file. QuickHelp returns control to the program only when you exit QuickHelp.

A.13 Searching for a Topic

You can search the database for a topic by choosing the Search command in the View menu. This command searches the database for the topic that has the name you supply. If it finds the topic, it displays the topic in the window. Otherwise, it displays the message “*topic* not found.”

To search for a topic, do the following:

1. Choose the Search command in the View menu.

QuickHelp displays a dialog box prompting you for the name of a topic.

2. Enter the topic name to search for and press the ENTER key. The name may be any combination of upper- and lowercase letters. QuickHelp does not use case when searching.

QuickHelp searches the database for the topic whose name you supply. The search includes the entire database, not just the current topic.

Since you may not always remember exactly how a topic name is spelled, QuickHelp will match the first letters of a partial name with the closest topic when it carries out a search. For example, if you supply the four letters “viog” for the search, QuickHelp uses the first three letters to select a category and uses the fourth letter to determine which function in the topic list should receive the highlight.

As you move from topic to topic, QuickHelp remembers your last position within each topic. If you return to a topic you have previously chosen in your QuickHelp session, QuickHelp displays the topic at its previous position. This means that the beginning of a topic may not appear in the window when you choose it.

Press S to choose the Search command without selecting the View menu.

A.14 Viewing Related Topics

You can quickly view topics related to the current topic by using the References menu. This menu displays the names of topics that are referenced by or related to the current topic. QuickHelp lets you choose a topic name from the list just as you would choose a command from any other menu. Choosing a topic name directs QuickHelp to display the topic.

To view a related topic, do the following:

1. Select the References menu.
2. Choose the name of the topic you want to view.

For example, if you are viewing the topic **KbdCharIn**, the References menu lists **KbdPeek** and **KbdStringIn**. To view the **KbdPeek** topic, choose it from the References menu. QuickHelp displays the topic immediately.

Figure A.4 shows the References menu for the **KbdCharIn** function:

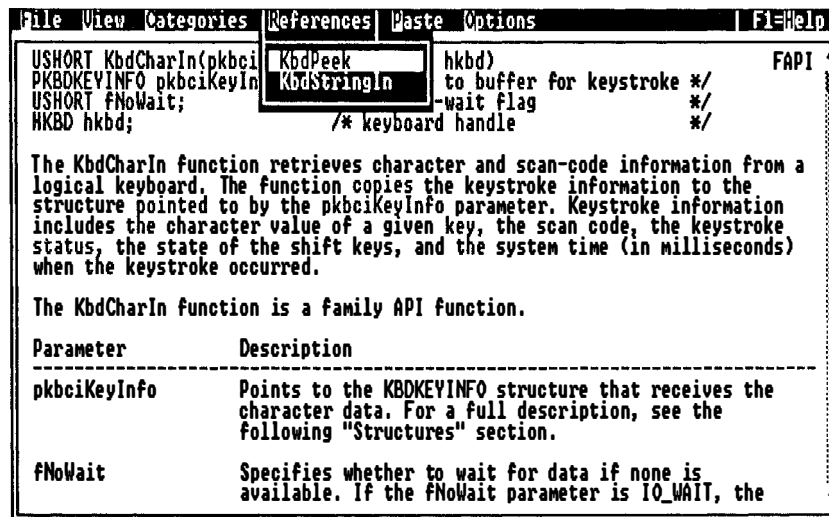


Figure A.4 KbdCharIn References Menu

If you choose the **KbdPeek** topic, you will see the window shown in Figure A.5:

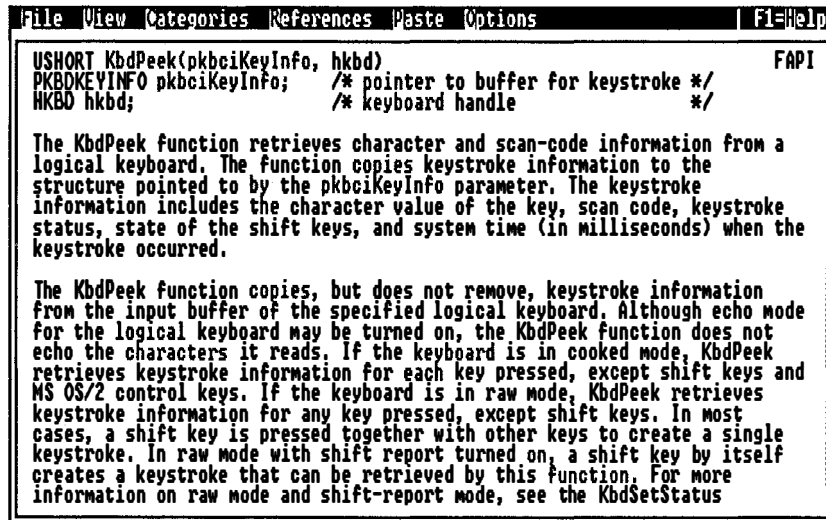


Figure A.5 KbdPeek Window

If the current topic has no references, the References menu lists the message “No References.”

A.14.1 Viewing the Next Topic

You can view the next topic in the database by choosing the View Next command in the View menu. This command displays the next topic in the database; for example, the next topic in the current category. QuickHelp continues to display the next topic until it reaches the end of the database.

The View Next command does not stop at the end of the current category. If you are displaying the last topic in a category and choose the View Next command, QuickHelp moves into the next category and begins displaying topics from that category. When QuickHelp reaches the end of the database it displays the message “You are at the end of the database.”

Press N to choose the View Next command without selecting the View menu.

A.14.2 Returning to the Last Topic

You can return to the topic you previously viewed by choosing the View Last command in the View menu. This command displays the topic that immediately preceded the current topic. For example, if your current topic is **KdbPeek** and your previous topic was **KdbCharIn**, choosing the View Last command displays the **KdbCharIn** topic.

You can also view the previous topic by clicking the right mouse button.

A.14.3 Viewing a History of Your Session

You can display a list of the last 40 topics you have viewed by choosing the View History command in the View menu. This command is similar to choosing a category from the Categories menu, but it displays a list of the topics you have viewed rather than the topics in a specific category.

QuickHelp keeps a record of the last 40 topics that you have viewed, in the order in which you viewed them. Whenever you view a topic, the name of the topic is added to the history list. When you view a topic more than once, QuickHelp places the topic name at the top of the history list and removes any other occurrence of the topic from the list.

Press I to choose the View History command without selecting the View menu.

A.15 Searching Within a Topic

You can search for names and other character strings within a topic by choosing the Go To command in the View menu. This command searches the text of a topic for a string that you supply. If it finds the string in the topic, it scrolls the topic so that the line containing the matching string is at the top of the window.

To search within a topic, do the following:

1. Choose the Go To command in the View menu.
QuickHelp displays a dialog box prompting you for a string.
2. Type the string to be searched for and press the ENTER key. The string can be any combination of upper- and lowercase letters. QuickHelp does not use case when searching.

QuickHelp searches for the string, starting at the next line, moving through to the end of the file, and continuing at the beginning of the file until it reaches the current line, if necessary.

You can also quickly search for structures sections and family API sections within a topic by choosing the Go To Structure commands and Go To Family API commands. These commands search the current topic for the corresponding section and scroll the topic so that the first line of the section is at the top of the window. A structures section defines the fields of structures that are used with a function. A family API section describes any restrictions you must observe when using the function in a real-mode program. If a topic does not have a structures section, QuickHelp displays the message “This topic has no structures section.” If a topic does not have a family API section, QuickHelp displays the message “This topic has no family API restrictions section.”

A.16 Pasting from the Database

You can copy all or portions of a topic from the database to the QuickHelp paste file by choosing a paste command from the Paste menu. You can then paste the contents of this file into a source file that you are editing. The paste commands are a quick way to copy program samples, function syntax, and structure definitions to your program.

The Paste menu has the following commands:

Command	Action
Current Window	Copies the contents of the window.
Current Topic	Copies the entire topic.
Syntax	Copies the syntax section. Applies to function topics only.
Structure	Copies the structures section. Applies to function topics only.
Example	Copies the example section. Applies to function topics only.

When you select this menu, QuickHelp displays a list of sections that are available for pasting. Once you select the section you want to paste—for example, the syntax section of a function description—the section is copied by default into a file named *paste.qh* in your *tmp* directory.

In addition to using the Paste menu, you can also specify paste operations from the Microsoft Editor. For more information about how to do this, see Section A.18.4, “Pasting from the Microsoft Editor.”

A.16.1 Setting the Paste-File Mode

You can set the paste-file mode by choosing the Append Paste File or Overwrite Paste File commands in the Options menu. The paste-file mode determines whether QuickHelp removes the current contents of the paste file before copying text to it, or whether it just appends the copied text to the end of the existing contents.

If the paste-file mode is overwrite, QuickHelp exits immediately after copying text to the paste file. If the paste file mode is append, QuickHelp does not exit immediately after writing to the paste file. This allows you to continue pasting to the file.

A.16.2 Renaming the Paste File

You can rename the paste file by choosing the Rename Paste File command in the File menu. This command sets the name of the paste file to the name you supply.

To rename the paste file, do the following:

1. Choose the Rename Paste File command in the File menu.
QuickHelp displays a dialog box prompting you for the new filename.
2. Enter the new paste filename and press the ENTER key, or press the ESCAPE key to cancel the command.

If you do not specify a full pathname, QuickHelp creates the paste file in the current directory. If you supply an invalid filename, QuickHelp displays an error message and prompts you for a new filename when you choose a paste command.

A.17 Opening a New Database

You can open a new database by choosing the Open Database command in the File menu. This command opens a database and displays the help topic in that database.

To open a new database, do the following:

1. Choose the Open Database command in the File menu.
QuickHelp displays a dialog box prompting you for the name of a database.

2. Enter the filename of the database you want to open and press the ENTER key. If you don't supply a full pathname, QuickHelp searches the directories in your PATH environment variable for the database.

If QuickHelp cannot find the database or the database is not valid, QuickHelp displays the message "Cannot open *filename*."

A.18 Using QuickHelp with the Microsoft Editor

You can use QuickHelp as an extension to the standard Microsoft Editor to give you immediate on-line help while editing your MS OS/2 program source files. The Microsoft Editor lets you select, view, and paste topics by using the *quickhelp* function and without leaving your editing session.

A.18.1 Installing QuickHelp in the Microsoft Editor

To use QuickHelp with the Microsoft Editor, you need to install the **qhmp** extension library *qhmp.dll* and assign a key to the *quickhelp* function. This extension library provides the interface you need to access QuickHelp and the database.

To install the **qhmp** extension library, use the *assign* function to assign the argument "load:qhmp". For example, the following command installs QuickHelp and displays the message "Press ALT+Q to access QuickHelp."

```
ALT+A load:qhmp ALT+=
```

If you want to install QuickHelp each time you start the Microsoft Editor, add the command **load:qhmp** to the [mep] section of your *tools.ini* file. This installs QuickHelp automatically when you start.

A.18.2 Opening the QuickHelp Window

You can open the QuickHelp window by using the *quickhelp* function. This function, whose default key assignment is ALT+Q, opens the QuickHelp window and displays a topic. Typically, QuickHelp displays the QuickHelp help topic, but can also display other topics, depending on whether you specify a search topic.

While the QuickHelp window is open, you may carry out any QuickHelp commands. When you have finished viewing topics, press X or choose the Exit QuickHelp command to return to your editing session.

If you are used to using the View History command in QuickHelp, it is important to note that QuickHelp discards your history list whenever you return to the Microsoft Editor.

A.18.3 Searching for a Topic from the Microsoft Editor

You can search for a topic by supplying a topic name as an argument to the *quickhelp* function. This directs QuickHelp to search for and display the topic when it starts.

To search for a topic, do the following:

1. Press ALT+A (the *arg* function) and enter the name of the topic you want to search for.
2. Move the cursor to the word in your source file that you want to search for.
3. Press ALT+Q (the *quickhelp* function) to search for the help topic.

If QuickHelp does not find the topic, it displays the message “*topic* not found.” To continue, press any key. QuickHelp then opens its window and displays the QuickHelp help topic.

A.18.4 Pasting from the Microsoft Editor

You can copy the syntax or example section for a given function from the MS OS/2 System Functions database into the paste file by using the *quickhelp* function. The *quickhelp* function then switches the editing session to the paste file, from which you can copy text to the pick buffer and return it to your original file.

To copy the syntax or example section of an MS OS/2 function into the paste file, do the following:

1. Move your cursor in your source file to the topic name.
2. Press F9, ALT+Q (the *meta* and *quickhelp* functions) to copy the syntax.
3. Press ALT+A, ALT+A, ALT+Q (the *arg* and *quickhelp* functions) to copy the example.

If the topic name does not appear in your source file, you can enter the topic name as an argument instead of moving your cursor. For example, the following command copies the syntax section of the **DosOpen** function to the paste file:

F9 ALT+A DosOpen ALT+Q

The following command copies the example section of the **DosOpen** function to the paste file:

ALT+A ALT+A DosOpen ALT+Q

Once you have copied text to the paste file, MEP switches the editing session to that file. You can then copy the text that you want to the pick buffer by using the *copy* function, and return to your source file to insert the pick buffer contents by using the *setfile* and *paste* functions. For example, to copy the complete example of the **DosOpen** function, enter the following command:

ALT+A ALT+A DosOpen ALT+Q ALT+A 15 CTRL+INS F2 SHIFT+INS

For more information about cut and paste operations that you can perform from the Microsoft Editor, see the *Microsoft Editor User's Guide*.

A.19 Specifying Parameters on the Command Line

You can tell QuickHelp which topic to display and what paste file to use by using the QuickHelp command-line parameters. You have the option of specifying the file to which the pasted text will go; you can specify whether to exit from QuickHelp after the paste operation completes, or whether data should overwrite or append the contents of files. You can also specify which section of a topic description to paste, and you can perform a paste operation directly, without displaying the QuickHelp window.

The QuickHelp command line has the following form:

qh [*topic*] [*options*]

The *topic* parameter is any valid QuickHelp topic. The *options* parameters are as follows:

Option	Description
-p <i>pathname</i>	Sets the name of the paste file. This option acts in the same way as the Rename Paste File command in the File menu. You do not need to specify a topic with the -p option. To send pasted text to the screen, you can specify con as the <i>pathname</i> . QuickHelp ends when the pasting is completed.

-pa [*pathname*] Changes the paste-file mode from the overwrite default mode to append. You do not need to specify a topic with the **-pa** option. If you specify a pathname, the **-pa** option changes the name of the paste file; otherwise, the text is pasted to the *paste.qh* file in your *tmp* directory.

-t *section* Directs QuickHelp to copy the specified section of the given topic to the current paste file. You must specify a topic with the **-t** option. If the paste-file mode is append, QuickHelp displays the specified topic in the window. If the paste-file mode is overwrite, QuickHelp ends immediately after copying the section to the paste file.

The *section* parameter can be one of following:

Section	Meaning
example	The example section of a function.
structure	The structures section of a function.
syntax	The syntax section of a function.
all	The entire topic description.

n Specifies the height of the window in lines. The height can be any number from 5 to the maximum height of the window.

+n Sets the maximum height of the screen in lines.

For example, the following command displays the **DosOpen** topic in the QuickHelp window:

```
qh DosOpen
```

The following example starts QuickHelp and names the paste file *test.1*:

```
qh -p test.1
```

The next example copies the syntax section of the **DosOpen** function to the screen:

```
qh DosOpen -p CON -t syntax
```

A.20 Using the QH Environment Variable

You can specify the default database, paste filename, paste-file mode, window height, and maximum screen height by setting the QH environment variable. QuickHelp reads the QH environment variable when it first starts and uses the values given with the variable as the default values.

To assign values to the QH variable, you need to use the **set** command and specify one or more arguments for the variable. The QH environment variable has the following form:

```
set qh=[[pathname...] [n] [-p pathname] [-pa [pathname]]
```

Parameter	Description
<i>pathname</i>	Specifies the pathname of the database you want to use. You can specify more than one database. By default, QuickHelp loads the <i>qh.hlp</i> database, which contains the MS OS/2 system functions.
<i>n</i>	Specifies the number of lines that the QuickHelp window will occupy.
-p <i>pathname</i>	Sends pasted text to the file specified by <i>pathname</i> . After the pasting is completed, QuickHelp ends.
-pa [<i>pathname</i>]	Appends pasted text to the file specified by <i>pathname</i> , but does not exit QuickHelp. If you do not specify a <i>pathname</i> , the text is pasted to the <i>paste.qh</i> file in your <i>tmp</i> directory.

For convenience, you can include the **set** command for the QH variable in your *os2init.cmd* file. This ensures that the variable is set for each invocation of QuickHelp.

Index

- .286 directive, 20
- 1024 bytes (-K) multiplier, 38

- A (address) option, 36
- A (ASCII) option, 36
- AbxA default option, 36, 38
- Address, 17, 21, 36
- All section parameter, 60
- ALT+=, 57
- ALT+A, arg function, 57-59
- ALT+Q, quickhelp function, 57-59
- /AM option, 14
- API (application programming interface) 13
- Append Paste File command, 56
- Arg function, 58
- ASCII format, 36, 38
- .Asm filename extension, 22
- /Asnu option, 16, 17
- Assembly language, 20-26
- Assign function, 57
- Asterisk (*), 38, 39

- B (byte) option, 36
- B (bytes) multiplier, 38
- Backslash (\), 37
- Base option, 36, 37
- BASIC compiler, 3
- Bind program, 3, 13
- Binding
 - applications, 13
 - messages to programs, 34-35
 - programs, 10
- Bold text, 5
- Brackets, double, 5
- Building
 - See also* Creating dynamic-link libraries
 - See* Dynamic-link library
 - large memory model programs, 13-14
 - programs, 9-14
- Byte (-b) option, 36
- Bytes (-b) multiplier, 38

- C (character) option, 36
- C library, 16
- C optimizing compiler, 3, 9, 12, 16

- C option, 11, 14, 17
- /C parameter, 31
- C source code *See* Source code
- C-language header file, 44
- C-language program, 9
- Calling program, 16
- Caret (^), 37
- Categories menu, 47
- Category, 43, 47
- Character
 - format, 36
 - option (-c), 36
 - special, 37
 - wildcard (*, ?), 39
- Choosing
 - categories, 47
 - QuickHelp commands, 46-47
 - topics 48-50
- Cl command, 11, 12, 16, 17
- /CO option, 12, 19, 20
- .CODE directive, 21
- CodeView debugger, 10, 12, 17, 20
- Colon (:), 33
- Command
 - See also specific command*
 - choosing, 46-47
- Comment line, 32
- Compact-model program, 13
- Compiler *See specific compiler*
- Compiling
 - dynamic-link libraries, 17
 - source code, 11
 - with linking, 11
 - without linking, 14
- Component ID line, 32
- Component message line, 32-33
- Config.sys file, 44
- Control character, 37
- Convention
 - notational, 5
 - Pascal calling, 22
- Copy function, 59
- Copying
 - example section, 43, 55
 - from database to QuickHelp, 55
 - structures section, 55
 - syntax section, 43, 55, 58-59
 - topics, 55
 - windows, 55
- CPU register, 26

- Creating
 - See also* Building
 - import libraries, 3, 18
 - message files, 3
 - module-definition files, 17
 - programs, 9
 - run-time libraries, 3
- Current Topic command, 55
- Current Window command, 55
-
- D (decimal) option, 37
- .DATA directive, 21
- Data pointer, setting, 17
- Data segment, 16, 17, 24-25
- Database, 43
 - default, 45
 - MS OS/2 System Functions, 43-44
 - opening, 56-57
 - qh.hlp, 61
- Date, displaying, 39
- Dcall program, 20
- Debugging
 - /CO option, 19
 - dynamic-link libraries, 20
 - programs, 10, 12
 - with Microsoft Macro Assembler, 3
- Decimal (-d) option, 37
- Decimal format, 36
- Default
 - base option setting, 36
 - database, 45
 - format option setting, 36
 - library, 11, 14
 - paste filename, 45
 - qh.hlp database, 61
- Delay parameter, 30
- Development environment, 9
- DGROUP segment, 17, 24
- Directives, 20-21
- Directory parameter, 39
- Diskcache statement, 44
- Displaying
 - ASCII characters, 38
 - command-line information, 10
 - files *See* File
 - help information, 3
 - memory segment contents, 36
 - messages, 32
 - parameters, 10
 - syntax, 10
 - topics, 43
- Dos functions, 43
- DosBeep function, 22
- Doscalls.lib library, 12, 14, 20
- DosExit function, 22
- DosGetMessage function, 32
- DosOpen function, 58-59, 60
- DosOpen topic, 60
- DosWrite function, 16, 22
- DOWN key, 47, 48, 49
- Dual-mode application, 3
- Dynamic-link library
 - assembly language, writing in, 22-25
 - building, 15-16
 - compiling, 17
 - creating module-definition files, 17
 - data segment, 24-25
 - debugging, 20
 - dynlink.dll, 18
 - export definition, 18
 - import definition, 18
 - LIBRARY statement, 11
 - linking, 19
 - Microsoft C Optimizing Compiler, 16
 - module-definition file requirement, 17
 - QuickHelp, 44
 - source-code editing, 16
 - stack setting, 17
- Dynamic-link module, 12
- Dynlink.dll (dynamic-link library), 18
-
- E (error) message type, 33
- /E parameter, 39
- Editing source code, 10, 16
- Editor, Microsoft *See* Microsoft Editor
- Ellipses (...), 5
- END key, 49
- End statement, 26
- Environment variable (QH), 61
- Environment, setting up, 9
- Error (E) message type, 33
- ESCAPE key, 47, 56
- Example command, 55
- Example section parameter, 60
- Executable file *See* File
- Exehdr program, 3
- Exit QuickHelp command, 46, 57
- Export definition, 18
- EXPORTS statement, 18
- Extension, filename, 22, 32
-
- F option, 37
- F1 key, QuickHelp information, 46
- F8 (trace) key, 20
- F9 key, 58
- Family API function, 13, 55
- Far address, 21

Far segment, 16

File

 .asm filename extension, 22

 C-language header file, 44

 config.sys, 44

 displaying

 contents, 3

 date, 3, 39

 file information, 39-40

 filename, 38

 location, 3

 time, 39

 type, 39

 executable file

 header, 3

 modifying, 34

 reading from, 34

 type display, 39

 extensions, 22, 32

 function definition file, 10

 infile parameter, 32, 34

 location, 3

 make description file, 14-15

 menu, 50, 56

 message, 3

 module-definition file *See* Module-
 definition file

 .msg filename extension, 32

 mshelp.dll, 44

 object file, 17, 19

 os2.h, 10, 14, 16

 os2def.h, 10

 os2init.cmd, 61

 outfile parameter, 32

 parameter, 39

 paste file, 56, 59, 60

 paste.qh, 55, 60

 qh.exe, 44

 qh.hlp, 44

 qhddl.dll, 44

 qhmp.dll, 57

 QuickHelp program, 44

 readme file, 16

 searching, 39-40

 size display, 39

 structure definition file, 10

 time display, 39

Filename extension, 22, 32

Format option, 36

-Format parameter, 36-37

Frame pointer, 24

Frame, stack, 23-25

Function

 Arg, 58-59

 Assign, 57

 Copy, 59

Function (*continued*)

 definition file, 10

 Dos, 43

 DosBeep, 22

 DosExit, 22

 DosGetMessage, 32

 DosOpen, 58-59, 60

 DosWrite, 16, 22

 Family API, 13

 Initialization, 26

 Input, 44

 IOctl, 44

 Kbd, 43

 KbdCharIn, 52

 Kernel, 43

 Keyboard, 43

 Mouse (Mou), 43

 MS OS/2 System Functions database
 43

 Output, 44

 Paste, 59

 Public, 22

 Quickhelp, 57, 58-59

 Sample, 18

 Setfile, 59

 Video (Vio), 43

/G2 option, 11

Global variable, 16

Go To command, 54

Go To Family API command, 55

Go To Structure command, 55

Greater-than (>) sign, 34

/Gs option, 17

H (help) message type, 33

H key, QuickHelp information, 46

Help

See also QuickHelp

 information, 3, 10

 library, 44

 messages, 33

Hexadecimal (-x) option, 37

Hexadecimal format, 36

Hide Window command, 51

HOME key, 49

Huge-model program, 13

I (information) message type, 33

I key, View History command, 54

Implib program, 3, 18-19

Import definition, 18

Import library, 3, 9, 11, 18

- IMPORTS statement, 18
- Infile parameter, 32, 34
- Information (I) message type, 33
- Initialization function, 26
- Input functions, 44
- Installing
 - See also* Setting up
 - QuickHelp, 44
 - QuickHelp into Microsoft Editor, 57
- IOctl functions, 44
- Italic text, 5

- K (1024 bytes) multiplier, 38
- Kbd function, 43
- KbdCharIn function, 52
- Kbdp program, 3, 29, 30
- KbdPeek topic, 53
- Kernel function, 43
- Key *See specific key*
- Keyboard
 - category choosing, 47
 - command choosing, 46
 - functions, 43
 - scrolling, 49
 - topic choosing, 48
 - typing rate, 3, 30

- L (long word) multiplier, 38
- L (long word) option, 36
- Large-model program, 13
- LEFT key, 47
- Less-than (<) sign, 34
- Lib program, 3, 11
- Libpath command, 16, 31
- LIBPATH directory, 44
- LIBRARY statement, 11, 18
- Library
 - C library, 16
 - debugging, 20
 - default, 11, 14
 - doscalls.lib, 12, 14
 - dynamic-link *See* Dynamic-link library
 - dynlink.dll (dynamic-link library), 18
 - import library, 3, 9, 11, 18
 - libwhere program, 31
 - linking with programs, 9, 11
 - location, 3, 31
 - mllibcep.lib, 14
 - parameter, 31
 - readme file, 16
 - run-time library, 3, 9, 12, 16
 - slibcep.lib, 14
- Libwhere program, 3, 29, 31

- Link program, 3, 11-12, 17-19
- Linking
 - defined, 11
 - dynamic-link libraries, 19
 - libraries with programs, 11
 - link program, 3, 11
 - programs, 11-12
 - programs with libraries, 9
 - run-time library, 12
 - with compiling, 11
- Load:qhmp argument, 57
- Loadds keyword, 16, 17
- Local variable, 16
- Long word (-l) multiplier, 38
- Long word (-l) option, 36

- M option, 37
- Macro Assembler, 3
- Macros, 44
- Make description file, 14-15
- Make program, 14-15
- Medium-model program, 13, 14
- Memory increasing, program, 13
- Menu *See specific menu*
- Message, 3, 32-34
- Microsoft
 - BASIC Compiler, 3
 - C Optimizing Compiler, 3, 9, 12, 16
 - CodeView debugger, 10, 12, 17, 20
 - Editor *See* Microsoft Editor
 - Help library, 44
 - Macro Assembler, 3
 - Pascal Compiler, 3
- Microsoft Editor, 10, 45, 57-59
- Mixed option, 20
- Mkmsgf program, 3, 29, 32-33, 34
- Mlibcep.lib library, 14
- Mode
 - default overriding, 37
 - dual, 3
 - paste-file, 56, 60
 - protected, 3, 10, 11, 13
 - real, 3, 10, 55
- Mode parameter, 37
- .MODEL directive, 21
- Model program, 13
- Model type, 21
- Module-definition file, 11, 17-19
- Monospace text, 5
- Mou functions, 43
- Mouse
 - category choosing, 47
 - command choosing, 47
 - functions (Mou), 43
 - scrolling, 49

Mouse (*continued*)

- topic choosing, 48, 49
- MS OS/2 program *See* Program
- MS OS/2 System Functions database
 - 43-44, 45
- .Msg filename extension, 32
- Msgbind program, 3, 29, 34-35
- Mshelp.dll file, 44
- Multiple-thread
 - debugging, 12
 - dynamic-link library, 16
- Multiplier, 38

- N count parameter, 38
- N key, View Next command, 53
- N option, 60
- +N option, 60
- N parameter, 61
- Name parameter, 36, 38
- Near address, 21
- Near data pointer, setting, 17
- No Default Library Search (/NOD)
 - option 12
- /NOD option, 12
- /NOI option, 19
- Notational conventions, 5

- O (octal) option, 37
- Object file, 17, 19
- Octal format, 36
- Octal (-o) option, 37
- Offset, 37, 38
- Open Database command, 56
- Option
 - See also* Parameter
 - a (address), 36
 - A (ASCII), 36
 - /AM, 14
 - /Asnu, 16, 17
 - b (byte), 36
 - base, 36, 37
 - c (character), 36
 - /c, 11, 14, 17
 - /CO, 12, 19, 20
 - d (decimal), 37
 - f, 37
 - format, 36
 - /G2, 11
 - /Gs, 17
 - hexadecimal (-x), 37
 - l (long word), 36
 - m, 37
 - Mixed, 20
 - n option, 60

Option (*continued*)

- +n, 60
- No Default Library Search (/NOD),
 - 12
- /NOD, 12
- /NOI, 19
- o (octal), 37
- p, 59
- pa, 60
- Remove Stack Probes (/Gs), 17
- t, 60
- t (text), 37
- View, 20
- w (word), 36
- x (hexadecimal), 37
- /Zi, 12, 17, 20
- /Zl, 17
- Options menu, 56
- OS/2 program *See* Program
- Os2.h file, 10, 14, 16
- Os2def.h file, 10
- Os2init.cmd file, 61
- Outfile parameter, 32
- Output functions, 44
- Overwrite Paste File command, 56

- P option, 59
- P parameter, 61
- P (prompt) message type, 33
- Pa option, 60
- Pa parameter, 61
- PAGE DOWN key, 49
- PAGE UP key, 49
- Parameter
 - See also* Option
 - all, 60
 - /c, 31
 - delay, 30
 - directory, 39
 - displaying, 10
 - /e, 39
 - format, 36-37
 - file, 39
 - infile, 32, 34
 - library, 31
 - mode, 37
 - n count, 38
 - name, 36, 38
 - outfile, 32
 - p, 61
 - pa, 61
 - pushing on stack, 25
 - /q, 39
 - QuickHelp specifications, 59-60
 - /r, 39

Parameter (*continued*)

- rate, 30
- s offset, 37-38
- /t, 39
- Pascal calling convention, 22
- Pascal compiler, 3
- Paste
 - See also* Pasting
 - filename, default, 45
 - function, 59
 - menu, 55
- Paste file, 56, 59, 60
- Paste.qh file, 55, 60
- Pasting
 - See also* Copying; Paste
 - from database, 55
 - from Microsoft Editor, 58-59
- Pathname parameter, 61
- Plus (+) sign, 50
- Pointer *See* Mouse
- Pointer, frame, 24
- Program
 - assembly-language, writing in, 20-22
 - automating development, 14
 - bind, 3, 13
 - binding
 - applications, 13
 - messages to, 34-35
 - programs, 10
 - C-language, 9
 - calling programs, 16
 - compact-model, 13
 - creating programs, 9
 - dcall, 20
 - debugging programs, 10, 12
 - exehdr, 3
 - huge-model, 13
 - implib, 3, 18-19
 - increasing memory, 13-14
 - kbdp, 3, 29, 30
 - large-model, 13
 - larger memory models, 13-14
 - lib, 3, 11
 - libwhere, 3, 29, 31
 - link, 3, 11
 - linking programs, 11-12
 - make, 9, 14-15
 - medium-model, 13, 14
 - memory increasing, 13
 - mkmsgf, 3, 29, 32-33, 34
 - models, 13
 - msgbind, 3, 29
 - QuickHelp, 3, 10
 - shd, 3, 29, 36-38
 - small-model, 13
 - test, 11, 12

Program (*continued*)

- trace, 4
- tracefmt, 4
- where program, 3, 29, 39-40
- writing example, 10
- Program development, automating, 14
- Prompt (P) message type, 33
- Protected-mode application, 3, 10, 11, 13
- Public function, 22
- /Q parameter, 39
- QH environment variable, 61
- Qh.exe file, 44
- Qh.hlp database, 61
- Qh.hlp file, 44
- Qhddl.dll file, 44
- Qhmep.dll file, 57
- Question mark (?), 39
- Quickhelp function, 57, 58, 58
- QuickHelp
 - See also* Help
 - automatic installation, 57
 - command, choosing, 46-47
 - defined, 3, 10
 - dynamic-link library, 44
 - environment variable (QH), 46
 - exiting, 46, 57
 - help topic, 45
 - installing, 44
 - menus, 46
 - parameter specification, 59-60
 - program file, 44
 - QH environment variable, 46
 - quitting, 46, 57
 - screen, topic choosing, 50
 - starting, 45
 - window, 45, 49, 50-51, 57-58
 - with Microsoft Editor, 57-59
- /R parameter, 39
- Rate parameter, 30
- Reading from executable file, 34
- Readme file, 16
- Real-mode application, 3, 10, 55
- References menu, 52
- Register, 17, 23-24, 25
- Remove Stack Probes (/Gs) option, 17
- Rename Paste File command, 56
- RET instruction, 26
- RIGHT key, 47
- Run-time library
 - lib program, 3
 - linking, 9, 12

Run-time library (*continued*)

Microsoft C Compiler, 16

S key, Search command, 52

S offset parameter, 37-38

Sample function, 18

Screen, 50

Scroll bar, 49

Scroll box, 49

Scrolling keys, 49

Search command, 51

Searching

family API sections, 55

files, 39-40

libraries, 3

structures sections, 55

topics, 43, 51, 58

within topics, 54-55

SEGMENTS statement, 14

Selecting *See* Choosing

Semicolon (;), 32

Set command, 61

Setfile function, 59

Setting

See also Setting up

addresses, 17

base option, 36

data pointer, 17

data segment, 17

format option, 36

high bit, 37

near data pointers, 17

paste-file mode, 56

stack segment, 17

window height, 50

Setting up

See also Setting

data segment, 24-25

development environment, 9

frame pointer, 24

stack frame, 23-25

Setup operation, 11

Shd program, 3, 29, 36-38

Single-thread dynamic-link library, 16

Size, file display, 39

Slibcep.lib library, 14

Small-model program, 13

Source code, 10, 11, 16

Stack, 22, 25, 26

.STACK directive, 21

Stack frame, 23-25

Stack segment, setting, 17

String, 16

Structure

command, 55

Structure (*continued*)

definition file, 10

MS OS/2 System Functions database
43

section

copying, 55

parameter, 60

searching, 55

Syntax

command, 55

copying, 58-59

displaying, 10

section

copying, 55

parameter, 60

System Functions database, 43, 45

-T option, 60

/T parameter, 39

-T (text) option, 37

Test program, 11, 12

Text format, 36

Text (-t) option, 37

Tilde (~), 37

Time, displaying, 39

Topic

choosing, 48, 49

copying, 55

displaying quickly, 44

help, 45

KbdPeek, 53

list, 47

loading quickly, 44

next topic, viewing, 53

related topics, viewing, 52

scrolling, 49

searching within, 54-55

viewing, 43, 46, 49

Trace key (F8), 20

Trace program, 4

Tracefmt program, 4

Typing rate, keyboard, 3

UP key, 47, 48, 49

USHORT data type, 10

Utility macros, 44

Video functions, 43

View History command, 54, 58

View Last command, 54

View menu, 51, 53-54

View Next command, 53

View option, 20

Viewing

- next topic, 53
- related topics, 52
- topics, 43, 49

Vio functions, 43

W (warning) message type, 33

-W (word) multiplier, 38

-W (word) option, 36

Warning (W) message type, 33

Where program, 3, 29, 39-40

Wildcard characters (*, ?), 39

Window Height command, 50

Window

- copying, 55

- height, 50, 60

- hiding, 51

- QuickHelp, 49, 57

Word (-w) multiplier, 38

Word (-w) option, 36

-X (hexadecimal) option, 37

/Zi option, 12, 17, 20

/Zl option, 17