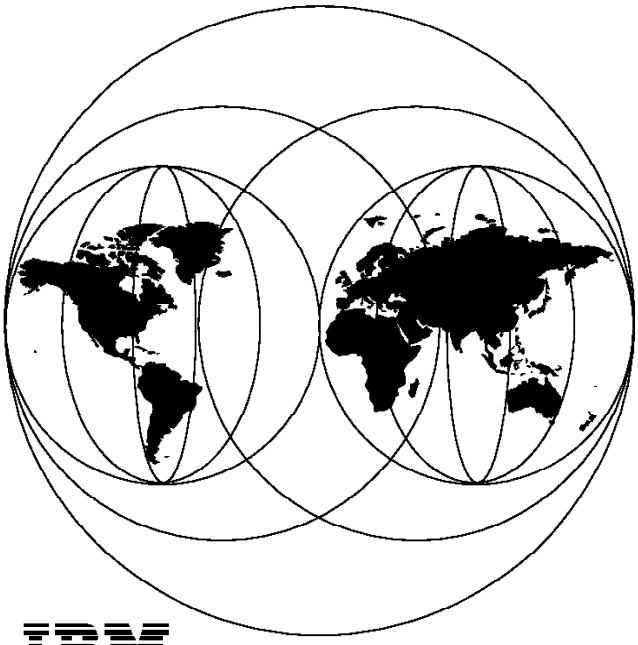
SG24-4719-00

Open32
Developer API Extensions for OS/2 Warp

December 1996



IBM

International Technical Support Organization
Austin Center

SG24-4719-00

International Technical Support Organization

Open32
Developer API Extensions for OS/2 Warp

December 1996

IBML

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix B, “Special Notices” on page 263.

First Edition (December 1996)

This edition applies to the Developer API Extensions of OS/2 Warp Version 3 with FixPak 17 or greater installed. The programs described in this edition will also execute on OS/2 Warp Version 4 utilizing the Open32 support included in OS/2 Warp Version 4.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. JN9B Building 045 Internal Zip 2834
11400 Burnet Road
Austin, Texas 78758-3493

Contents

Figures

Tables

Preface

This redbook is intended to provide an overview of developing C/C++ applications which utilize Open32 for OS/2 Warp. It describes techniques for building applications which utilize the same common source code that can be compiled and executed on both the OS/2 or Win32 platforms.

In addition to the discussion of developing new applications utilizing Open32, we explore porting existing Windows 32-bit applications and Windows 16-bit applications to the OS/2 environment. Tools that are available to simplify this process are also covered.

Sample programs are provided to illustrate techniques and procedures for writing programs utilizing Open32. Some samples show applications that contain only common source code that will execute on either the OS/2 Warp or Win32 environments, while others are constructed using both common source code and platform dependent source code sections for the OS/2 and Win32 environments.

All the files necessary to build and execute the sample applications covered in this redbook for both the OS/2 and Win32 environments are provided on the CD-ROM that is included with this redbook. Also on the CD-ROM is a BookManager version of this redbook so you can view the redbook electronically.

The sample programs described in this redbook will execute on any OS/2 Warp Version 3 system with Developer API Extensions support from The Developer Connection for OS/2 Volume 10 along with FixPak 17 or higher installed. These programs will also execute on any OS/2 Warp Version 4 system which has the Developer API Extensions support built in and is known as Open32.

This redbook is intended for application development specialists, and system technical specialists, including IBM customers, business partners, BESTeam members, system engineers, consultants, and independent software vendors who are interested in understanding the basic concepts for developing C/C++ applications that utilize Open32 for OS/2 Warp.

Some knowledge of OS/2 C/C++ and the Windows 32-bit applications is assumed.

How This Redbook Is Organized

This redbook contains 276 pages. It is organized as follows:

- Chapter 1, “Open32 Overview”

  This chapter provides a general overview of Open32 including:

  - Need for Open32 in OS/2
  - Architecture and contents
  - Benefits for application developers
  - Application development process using Open32
  - Tools available to aid the application development process

- Chapter 2, “Tools Used with OS/2 Developer API Extensions”

  This chapter discusses the tools that can be utilized to develop Open32 applications as well as tools that are available to aid in the porting of Windows 16-bit applications and Windows 32-bit applications to the OS/2 environment. Topics covered include:

  - Where to obtain the tools
  - How to install and configure the tools
  - How to get started with the tools

- Chapter 3, “Howdy, World!”

  This chapter describes the common steps in the process of application development using Open32. A simple application example will be discussed. This simple program will step by step be enhanced showing how standard Win32 resources of menu, accelerators and dialog boxes added to the program.

- Chapter 4, “MDI Sample Program”

  This chapter presents the development of a Multiple Document Interface program using Open32 functions. The child windows of the program are capable of presenting bitmap, graphic, screen capture or text data.

- Chapter 5, “Mixed Mode Sample Program”

  This chapter presents the programming technique of separating the common application code from operating system dependent code where the Win32 APIs used by the application are not supported by Open32. The two separate operating system dependent sections use equivalent OS/2 and Win32 APIs to perform the same function for the application.

- Chapter 6, “Named Pipe Sample Program”

  This chapter explores the programming technique of developing OS/2 function prototypes to provide missing Win32 functions of Open32. This technique then allows for the porting of a Win32 program to OS/2 without dividing the source code into platform dependent sections as described in Chapter 5, “Mixed Mode Sample Program” on page 133.

- Chapter 7, “Tree View Control Sample Program”

  This chapter presents the programming technique of providing a translation layer between the Win32 program and OS/2 functions for the Windows 95 common controls not supported by Open32. A Tree View function is implemented as an example of how this technique can minimize the amount of coding changes required to port a Win32 application to Open32.

- Chapter 8, “Existing Windows 16-bit Application Ported to OS/2”

  This chapter shows the steps required to migrate an existing Windows 16-bit application to the OS/2 environment and how Open32 simplifies the migration.

- Chapter 9, “Hints and Tips for Open32”

  This chapter presents several items to consider when developing Open32 applications based on the experiences of the authors of this redbook.

- Appendix A, “Common Problems and Easy Solutions”

  This appendix lists the more common programming missteps and mistakes that the authors of this redbook felt could be encountered by others working with the Open32 sample programs provided in this redbook. For each problem listed a solution is provided.

The Team That Wrote This Redbook

This publication was produced by a team of specialists from around the world working at the International Technical Support Organization Centers in Austin and Boca Raton.

The project was designed and managed by:

Mike Foster
International Technical Support Organization, Austin Center

Doris Corel
International Technical Support Organization, Boca Raton Center

The authors of this redbook are:

Theo Foster
International Technical Support Organization, Austin Center

Osamu Takagiwa
IBM Japan

Franck Yu
IBM France

Thanks to the following people for the invaluable advice and guidance provided in the production of this redbook:

Alex Gergor
International Technical Support Organization, Austin Center

Dan Mendrala
IBM Development, Austin

Dave Reich
IBM Development, Austin

Alex Tarpinian
IBM Development, Austin

Comments Welcome

We want our redbooks to be as helpful as possible. Should you have any comments about this or other redbooks, please send us a note at the following address:

redbook@vnet.ibm.com

Your comments are important to us!

Chapter 1. Open32 Overview

Today's PC technologies are changing very fast, introducing new products and creating new markets. This complicates the already difficult task of developing applications.

Some of the challenges currently facing software developers as a result of the quickly changing technologies include:

- Moving from 16-bit applications to 32-bit applications to follow the current market trend toward 32-bit platforms. This phenomena is being accelerated by the appearance of Windows 95.
- Keeping track of the emerging technologies, such as object-oriented programming languages, methods, tools and components.
- Adopting new techniques to optimize development investment, reduce costs and reduce the software delivery time.

With these challenges, software developers are asking themselves questions such as: What platform should we support? What object-oriented technologies should we choose? These are not always easy decisions to make. Software developers can choose to develop applications for only one platform. While this decision minimizes development and maintenance cost, it limits the product's market opportunity, as well as the choices of platforms the customers may consider.

On the other hand, they can choose to support multiple platforms. This option enlarges their product's market while at the same time removing the product's dependency on the success of any one platform. But it takes more resource to develop, test and maintain these multiple platform applications.

As for technologies, object-oriented technologies provide powerful tools for application development and component reuse, but application developers have a significant investment in procedural applications and development tools. The transition from procedural to object-oriented code will cost in terms of money and time.

IBM understands all these concerns and is working hard to be an open systems provider and to offer application developers cost effective solutions to create applications that will work across a variety of products, including hardware and software. The objectives of IBM's solutions for application developers are to:

- Provide tools and system components that reduce development, test and maintenance costs
- Leverage a common code base across multiple platforms
- Develop open industry standards, architectures, and parts
- Build upon advanced technologies

To accomplish these objectives, IBM provides the following products:

1. VisualAge family suite, a multiple platform tool set including:
   - C/C++, SmallTalk and Cobol compilers
   - Open Class Library
   - VisualBuilder
   - Data Access Builder
   - Analysis, test and debugging tools
2. OpenDoc, a multiple platform compound document architecture that enables the development of object-oriented multiple platform application components called OpenDoc parts.
3. Open32 that expands OS/2 by providing a subset of the Win32 APIs on OS/2 for source code compatibility, enabling the migration of Windows applications to OS/2.
4. Source Migration Analysis Reporting Toolset (SMART) tool that automates many of the tasks associated with migrating procedural 16-bit and 32-bit Windows code as well as 16-bit OS/2 code to 32-bit OS/2 code.
5. Hyperwise, a what-you-see-is-what-you-get (WYSIWYG) editor that enables the authoring of hypertext on-line information and application help for ø and Windows.

IBM is providing multiple platform solutions for application developers such that they can spend their resources writing new functions that will make their applications the best in the market place, while minimizing their efforts on non-development activities. For those developers who wish to work with the object-oriented technologies, the IBM VisualAge family product suite with Open Class Library and OpenDoc can provide a solution.

For those developers who want to continue to develop procedural code or need to maintain their investment in procedural code applications, Open32 provides a means to expand these applications to multiple platforms. The focus of this book is to provide technical information about Open32 and the

tools that can be used to develop applications that incorporate these extensions.

1.1 Why Open32?

For years, OS/2 has proven to be a secure, highly reliable operating system for mission-critical and enterprise applications. However, in spite of its strengths, OS/2 currently has less native PC applications written for it than those for Windows. Part of the reasons for the lack of native OS/2 applications is not OS/2 itself, but the fact that it is expensive to rewrite code and later maintain it for multiple platforms.

Most of existing Windows applications are written in a procedural programming language, the C/C++ language for example, and the component reuse is quite limited: porting a Windows application to OS/2 usually leads to a rewrite of the majority of modules. Even if it were done, to keep the function consistent, the code maintenance for Windows and OS/2 would not be trivial. If application developers can leverage their investment in Win32 application development by creating functionally equivalent native OS/2 applications at a small additional cost, there will be more native applications on OS/2. Applications developers will benefit and, consequently, OS/2 users will too.

Recognizing this fact, IBM has designed Open32 to ease the task of migrating existing Windows applications to OS/2. Open32 provides a source code level portability with the commonly used Win32 APIs. Windows applications that conform to Open32 can be recompiled to produce a functionally equivalent OS/2 application. The application then runs on OS/2 and has the look and feel of OS/2. It continues to be available as a Win32 application for Windows 95 and Windows NT.

1.2 Open32 Architecture

As shown in Figure 1 on page 4, Open32 is not Win32 APIs emulation. They are native OS/2 functions, being built right into the operating system, just like the other interface layers of OS/2: Control Program (DOS), Control Program (DOS), Presentation Manager (PM) and Graphics (GPI).

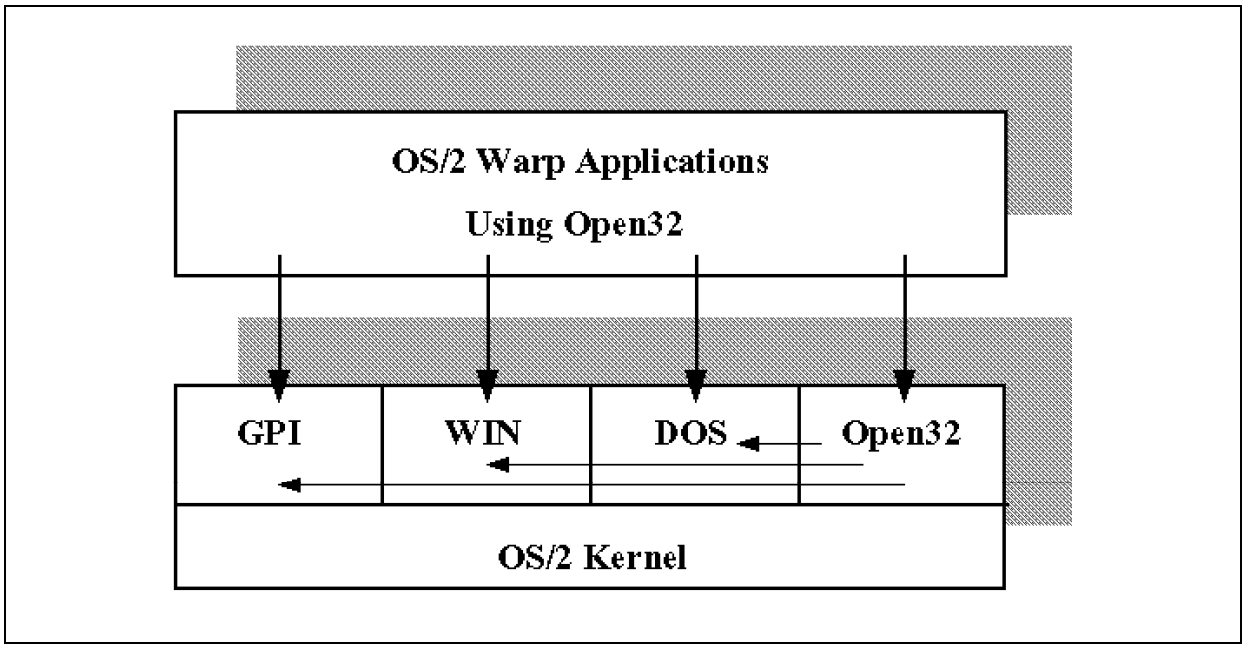


Figure 1. Open32 Architecture

For some historical reasons, OS/2 and Win32 APIs have many similarities either in their syntax and semantics. In many instances, Open32 APIs are simply wrappers around a similar OS/2 API: when called, they just pass the control to their OS/2 counterpart to do the job, eventually convert the input and output arguments and return values when necessary. In the other cases, extensive code needed to be written underneath the API, and within OS/2 itself, to perform the required function. However, all these implementation details are hidden from applications.

1.3 What is Open32?

Open32 are new application programming interfaces (APIs) added to the OS/2 Warp operating system. These new APIs provide an identical interface as the corresponding function in the Win32 environment does. The most commonly used Win32 functions and messages are implemented in Open32 with over 800 Win32 functions and almost all the Win32 messages. Following are the areas covered by Open32:

1. Base Operating System Services
   - Atoms
   - Date and time
   - Environment
   - Memory management

- Module management
- Printing
- Processes and threads
- Registry
- Resources
- File management

2. Common Dialog Boxes
   - Color and font selection
   - Opening and closing files
   - Printing
   - Text find and replace
3. Graphics Device Interface
   - Bitmaps, brushes, pens
   - Colors and color palettes
   - Fonts and text
   - Lines, curves, rectangles, polygons, ellipses, chords
   - Paths, regions
   - Mapping modes, transformations, coordinates
4. Window Management
   - Accelerators, carets, cursor, icons, menus, string tables
   - Main and Child windows
   - Dialog boxes
   - Buttons, Combo boxes, edit controls, list boxes, scroll bars
   - Messages and message queues
   - Rectangles
   - Timers
   - Dynamic Data Exchange Management Library (DDEML)
   - Multiple Document Interface (MDI)

The choice of these APIs is not arbitrary. It is based on a thorough analysis of more than nine million lines of source code from a variety of shipping Windows applications, along with input from Independent Software Vendors.

1.4 Independent Software Vendors Benefits

The benefits that Open32 represents for independent software vendors are numerous.

For Windows application developers, Open32 gives them access to the OS/2 market with reduced entry cost while staying on Windows market. Their applications will benefit from the maturity and robustness of OS/2 and all the features and functions available on OS/2, such as Workplace Shell Classes, the System Object Model (SOM) and OpenDoc.

For application developers who have separated source code bases for OS/2 and Windows, Open32 allows them to merge their code into a common source code base to reduce the application development, test and maintenance costs.

For OS/2 application developers, they will now have more functions at their disposal. Open32 can provide a means to port their applications to Windows 95 and Windows NT.

IBM believes that most application developers will realize an 80% common code solution for Windows and OS/2 Warp with the remaining 20% customized to exploit the unique features of each operating environment.

1.5 Tools

Writing applications that utilize advanced graphical user interfaces is both complicated and time consuming. Luckily, IBM and other venders provide tools which can aid in the application development process. In this section we will discuss some tools which you can use to expedite or automate activities associated with the porting or developing Open32 applications.

1.5.1 OS/2 Warp Toolkit

The OS/2 Warp Toolkit is your main resource for information on OS/2 program development. The toolkit provides you with sample applications, on-line reference books, and programming tools that make OS/2 programming both easier and faster.

The included sample applications demonstrate how to use almost every part of the complete OS/2 API. A SOM example shows you how to create new WorkPlace Shell objects that merge with the operating system and the user interface. Such objects can be created even for DOS and Windows programs to allow application integration with OS/2, even without reprogramming.

OS/2 Warp Toolkit also includes tools to manipulate resources such as icons, bitmaps, and fonts. In addition to the regular Icon Editor, Font Editor, and Dialog Editor, with the OS/2 Warp Toolkit there is now also the new Universal Resource Editor (URE). URE is a functionally rich tool which allows you to create and modify many kinds of resources. URE operates much like the resource editors available for Windows.

1.5.2 VisualAge C++

VisualAge C++ is the standard C/C++ compiler for OS/2. The compiler includes many additional tools to improve the performance and quality of your application. Advanced code optimization makes sure that your programs run as fast as possible with the smallest executable size.

A new linker, ILINK, improves the link time by removing the need for a prelink step. Additionally, it has a new free-form command line, allowing you to simply specify files while it figures out how they work together during linking. For backwards compatibility with LINK386, there is a parameter to allow the old five-part input file specification.

VisualAge C++ also includes a PM-based interactive debugger with full support for debugging multiple threaded applications. No bugs stand a chance against this powerful debugger. Full support for message queue spying and memory peeking is included.

1.5.3 SMART

The Source Migration Analysis Reporting Tool provides general support for Windows programmers looking to port their applications to OS/2. SMART can create custom OS/2 program templates from Windows programs, allowing the programmer to focus on rewriting the interface for OS/2

When using Open32, however, there is no need to use SMART to port the original Windows code to OS/2 because Open32 will allow it to compile for OS/2 without modification. As a result, you will only use the SMART tool to convert resources. While you could migrate with Open32 without using SMART, it is not recommended. Manually translating your resource files would be tedious and time wasting. Additionally, you would have to find a tool to convert your Windows icons, cursors, and bitmaps to OS/2 format. SMART handles all of this automatically for you.

1.5.4 Hyperwise

Hyperwise is an IBM product which helps you maintain help files across several platforms. You can import your current help file or create a new help document. Hyperwise has advanced editing tools which make it easy to create links between the pages of your help document.

Hyperwise can import files from the RTF and IPF formats, so applications which currently have Windows or OS/2 help files can migrate to Hyperwise for future maintenance.

Hyperwise also provides IPF for Windows, which allows you to use your OS/2 help files with your Windows application.

1.6 Application Design Considerations

The application design considerations for Open32 applications include all of the traditional application program design considerations for applications targeted for execution on any single operating system platform. In addition Open32 applications because they can be compiled and executed on the three different environments of OS/2, Windows 95 and Windows NT have additional considerations. Many of these additional considerations arise from the fact that the three operating environments are different with features and enhancements that are unique to each.

This section will cover those additional factors that were taken into consideration in the sample programs developed to illustrate Open32 through out this book.

1.6.1 Common versus Mixed Mode Code

The benefit to application developers of using Open32 when coding their applications is the ability to use the code they write on the OS/2, Windows 95 and Windows NT platforms. When the application is designed and coded so that the C/C++ code can be compiled without change when moving it between platforms can greatly reduce the expense associated with porting an application between platforms as well as reduce the time required to make the move. For this reason you want to try to design your application to use only those Win32 APIs that are supported in all the operating environments where you intend to execute your application.

Applications and programs that utilize only those Win32 APIs which are supported on all three platforms can be classified as being Common Code programs. The HOWDY.C sample program discussed in Chapter 3, “Howdy, World!” on page 79 and the MDI sample program in Chapter 4, “MDI Sample Program” on page 103 are examples of a Common Code programs where the source can be compiled and executed on both the OS/2 and Win32 platforms without change to the source code of the program.

Because of the requirements of any one application, it may not always be possible to use only the Win32 APIs available on all three platforms. Thus you will need to use other programming techniques to minimize the amount of source code that is not common between the operating system versions

of your application. The objective of the design that you select should be to develop a program structure and code base that will minimize the activity needed to migrate and maintain the application for each of the operating system environments. Applications that have slightly different source code depending on the operating system environment where they will be compiled and executed can be classified as mixed mode programs.

Following is a list of three sample programs that are discussed in this redbook each of which used a different programming technique for writing programs that utilize Win32 APIs which are not supported in Open32.

- MIXMODE.C in Chapter 5, “Mixed Mode Sample Program” on page 133

  The MIXMODE.C sample uses the technique of separating the common code from the operating system dependent code where the Win32 window classes used by the application are not supported by Open32. This technique works where there are equivalent OS/2 native window classes that provide the same function as the unsupported Win32 APIs in Open32.

- SERVER.C and CLIENT.C in Chapter 6, “Named Pipe Sample Program” on page 179

  The SERVER.C and CLIENT.C samples use the technique of creating OS/2 library and header files which implement native OS/2 API's for Win32 API calls that are not supported by Open32. The advantage of this technique over the technique used in the mixed mode sample is the Win32 source code remains unchanged between the Win32 and the OS/2 versions.

- TVTEST.C in 7.1, “How the OS/2 Tree View Control Works” on page 196

  The TVTEST.C sample uses the technique of creating OS/2 library and header files which implement native OS/2 window classes for the Win32 window classes that are not supported by Open32. The advantage of this technique as with the technique used in the named pipe sample as compared to the technique used in the mixed mode sample is that the Win32 source code remains unchanged between the Win32 and the OS/2 versions.

1.6.2 New verses Existing Code

If you are starting a new project, you can plan your use of features and functions available for all of the operating system environments where your application will be executed. If your goal is to deliver code that has 100 percent or almost 100 percent portability between the different operating system environments, you can take this into consideration in your application design. By selecting and implementing those features and functions that exist on all platforms you can maximize the use of common

code within the application. This was the approach used in the development of the HOWDY sample program described in Chapter 3, “Howdy, World!” on page 79.

There will be times when you will want to take an existing application and migrate it to the Open32 environment to expand its platform coverage. The original application could be either an OS/2 or Windows application. But, because of the fact that Open32 is a Win32 API implementation the natural migration will be from the Windows environment to the OS/2 platform. Migration of an existing Windows application to Open32 is described in the sample in Chapter 8, “Existing Windows 16-bit Application Ported to OS/2” on page 235.

1.6.3 How much can be Shared?

At some point in your application development process you may find that there are differences between the operating environments which does not allow for the the complete migration of your source code between platforms. You need to be aware of these in your application design so that you can make the correct design decisions for your application. Following are the differences that were noted during the development of the sample programs to illustrate Open32 programming throughout this redbook.

1.6.3.1 What is Not Supported?

Open32 is a subset of Win32 APIs. Open32 is targeted at the core operating system functions of Windows 95 and Windows NT.

Some of base Win32 APIs that are not currently supported are:

- Exception Handling
- Mailslots
- Networks
- OLE
- Pipes
- Plug and Play
- Security

Some of Win32 APIs operating system extensions that are not currently supported are:

- Multimedia
- MAPI
- Pen

- TAPI

The Common Controls Library, introduced in Windows 95 and NT V3.5.1 is not supported including controls such as:

- Image list
- List view and tree view
- Property sheet and tab control
- Rich edit control
- Up-down control, trackbar, progress bar and animation control
- Toolbar and tooltip control

1.6.3.2 What is Different?

Due to underlying differences in OS/2 and Win32 platforms, certain Open32 functions may have different behaviors when performed in Win32 and OS/2. For example:

- The maximum coordinates allowed in Win32 are not the same as those allowed in OS/2. Table 1 lists the maximums for both Win32 and OS/2.

Table 1. Maximum Coordinates Allowed

SPACE	Win32	OS/2
World/Page	-2(31) to 2(31)-1	-2(27) to 2(27)-1
Device	-2(27) to 2(27)-1	-2(15) to 2(15)-1

- Open32 provides registry function calls, but data storage and retrieval from the registry requires minor modifications to your code so it can work on multiple platforms.
- Open32 resource calls are not source compatible with Win32. Except for LoadResource, which returns a pointer to an OS/2 resource structure, each API takes a pointer to a resource structure as a parameter. These functions take/return pointers to OS/2 resource structures - not Win32 resource structures.
- Open32 and OS/2 handles are not always exchangeable. It is recommended that you do not mix them. In other words, you should not pass handles that are obtained from Open32 to OS/2 APIs and vice versa. An exception for this rule is you can create an OS/2 child window from a Open32 window or vice versa. You can see an example of this in the mixed mode sample program discussed in Chapter 5, “Mixed Mode Sample Program” on page 133. You will see it in a sample program.

- It is recommended that you do not mix Open32 and OS/2 functional area calls. For example, do not mix Open32 and OS/2 graphics (GDI) calls.

1.6.3.3 What is the Solution?

The limitations covered in 1.6.3.1, “What is Not Supported?” on page 10 and 1.6.3.2, “What is Different?” on page 11 are not new for Windows application developers. The same type of problems occur when they try to develop an application to run on both Windows 95 and Windows NT. Some Win32 APIs are available on Windows 95 but not on Windows NT, and vice versa, since parts of the kernel of Windows 95 are still 16-bit. The parameters of many Win32 APIs, although 32-bit, are actually used as 16-bit. Things can become even more complicated when you want to keep compatibility with Windows 3.x since you must be restricted to Win32s, a subset of Win32 APIs.

Fortunately many of the unsupported Win32 APIs and common controls have an equivalent API or class in OS/2 or other IBM products based on OS/2. For example, you find Pipes in OS/2, Mailslots in LAN Server and OLE support in OpenDoc. You have OS/2 NoteBook control for Win32 Tab control, OS/2 Container control for Win32 List view and Tree view, OS/2 Spin button for Win32 Up-down control, and OS/2 Slider control for Win32 trackbar.

If you design your application with these differences in mind you can build what is called a mixed mode application that has a large portion of its logic in a common code segment that uses Open32 APIs along with a platform specific section that takes advantage of the equivalent APIs or classes for both the OS/2 and Windows platforms.

1.7 Overview of Scenarios

The major part of this redbook will discuss the development of different Open32 application programs. Covered first will be applications where the source code files can be shared between Win32 and OS/2 development environments starting with Chapter 3, “Howdy, World!” on page 79. Programs that share the source between the Win32 and OS/2 environments are know as common source applications.

In Chapter 5, “Mixed Mode Sample Program” on page 133 we will examine the development of a mixed mode application to address the use of similar but different OS/2 and Win32 APIs in a single application. In Chapter 6, “Named Pipe Sample Program” on page 179 and Chapter 7, “Tree View Control Sample Program” on page 195 two approaches to developing interfaces or translation code to support Win32 APIs not supported in Open32. The objective of these techniques is to maintain common source

code for the application program instead of using the mixed mode technique discussed in Chapter 5, “Mixed Mode Sample Program” on page 133.

In Chapter 8, “Existing Windows 16-bit Application Ported to OS/2” on page 235 we discuss the steps needed to migrate an existing Windows 3.1 16-bit application to Open32 so the program can run as a native 32-bit application on OS/2 Warp.

Other topics covered in this book include a description of the tools used to develop the Open32 programs presented in this redbook. This is covered in Chapter 2, “Tools Used with OS/2 Developer API Extensions” on page 15 along with directions for installing these tools from the The Developer Connection for OS/2 Volume 10 CD-ROM. In Chapter 9, “Hints and Tips for Open32” on page 251 we include items you may want to consider when developing you own Open32 applications.

Chapter 2. Tools Used with OS/2 Developer API Extensions

This chapter discusses the tools used in developing, porting and maintaining Open32 applications using The Developer Connection for OS/2 Volume 10.

The Developer Connection for OS/2 contains a series of CD-ROMs. In The Developer Connection for OS/2 Volume 10, Disk 1 through Disk 4 are for OS/2, while additional CDs are for other platforms.

The following list briefly describes the tools required for application development with Developer API Extensions. They are listed in the order of recommended installation.

Tool	Description
FixPak 17 (XR_W017)	An IBM-supplied update for OS/2 Warp and OS/2 Warp Connect.
IBM Developer's Toolkit for OS/2 Warp	IBM's developer toolkit for OS/2 Warp and OS/2 Warp Connect.
SMART	A program from One Up Corporation that will automate part of the conversion process from Windows to OS/2.
VisualAge C++	IBM's premier C/C++ compiler for OS/2.
Developer API Extensions (run time)	The run-time libraries needed by OS/2 Warp and OS/2 Warp Connect to run Developer API Extensions applications.

2.1 The Developer Connection for OS/2 Volume 10

You can install a complete set of tools that can be used to develop OS/2 applications using Developer API Extensions from The Developer Connection for OS/2 Volume 10. Before you can install Developer API Extensions or any other tools from The Developer Connection for OS/2 Volume 10 you will first need to install The Developer Connection for OS/2 Volume 10. See 2.1.1, “Installing The Developer Connection for OS/2 Volume 10” on page 16 which discusses the steps involved in installing The Developer Connection for OS/2 Volume 10.

If you have installed an earlier edition of The Developer Connection for OS/2, you will need to install The Developer Connection for OS/2 Volume 10 before proceeding with the installation of any tools from The Developer Connection for OS/2 Volume 10.

2.1.1 Installing The Developer Connection for OS/2 Volume 10

The following instructions will guide you through The Developer Connection for OS/2 Volume 10 installation.

1. Put DISC 1 of The Developer Connection for OS/2 Volume 10 in your CD-ROM drive.
2. Open an OS/2 full-screen or windowed session.
3. Change the current drive to your CD-ROM, for example `E:`, as shown on the first command line of Figure 2.
4. Execute `DEVCON`, as shown on the second command line of Figure 2.

```
[C:\]E:

[E:\]DEVCON
```

Figure 2. Installation Command for The Developer Connection for OS/2 Volume 10

5. The Developer Connection for OS/2 Volume 10 installation program will display the installation screen as shown in Figure 3 on page 17.

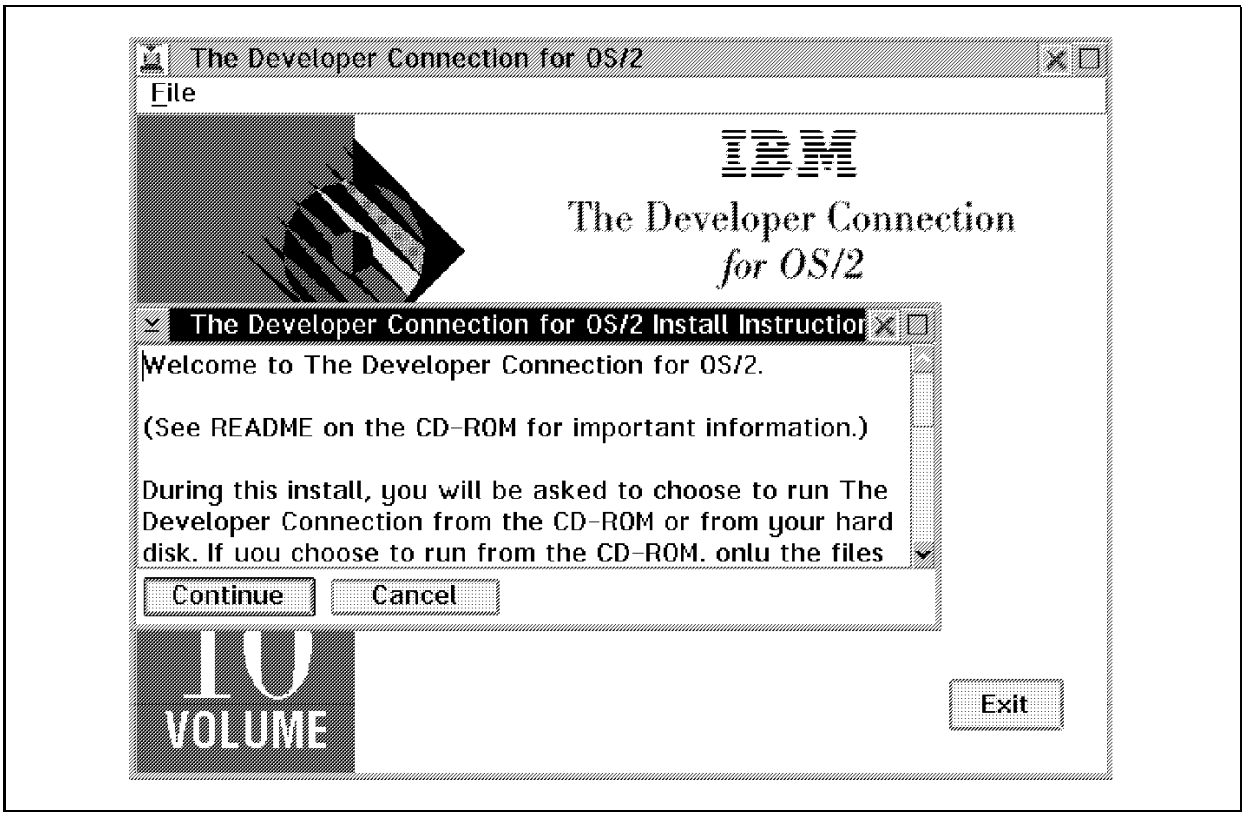


Figure 3. The Developer Connection for OS/2 Volume 10 Installation Window

6. After reading The Developer Connection for OS/2 install instructions, select **Continue** to proceed to the next screen as shown in Figure 5 on page 18.

Note

If you have previously installed any volume of The Developer Connection for OS/2, you will see the screen shown in Figure 4 on page 18. If you see this screen, select **Update the currently installed components** and press **Continue**. You can then skip steps 7 through 9.

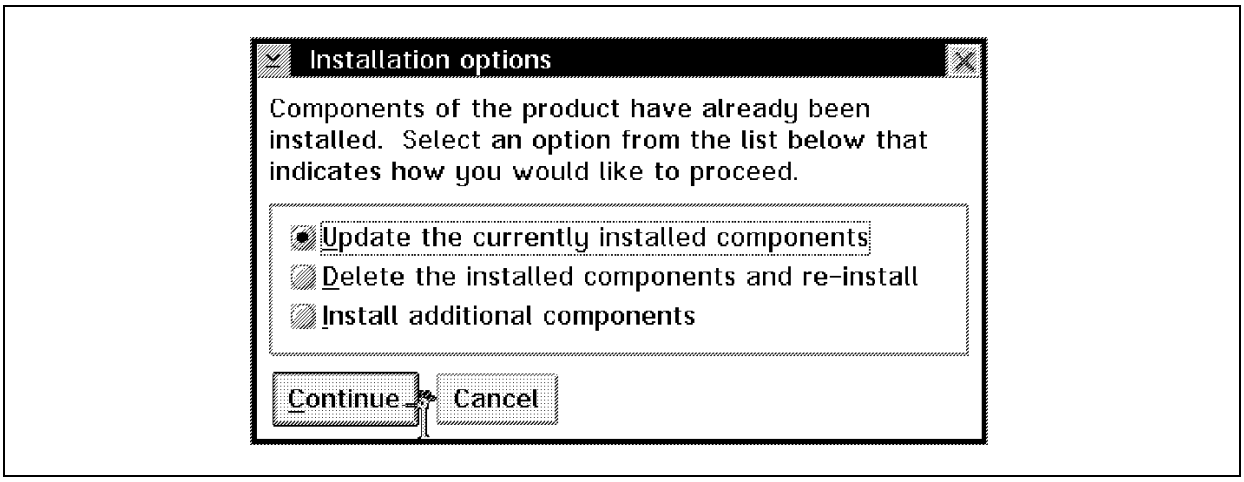


Figure 4. The Developer Connection for OS/2 Volume 10 Installation Option

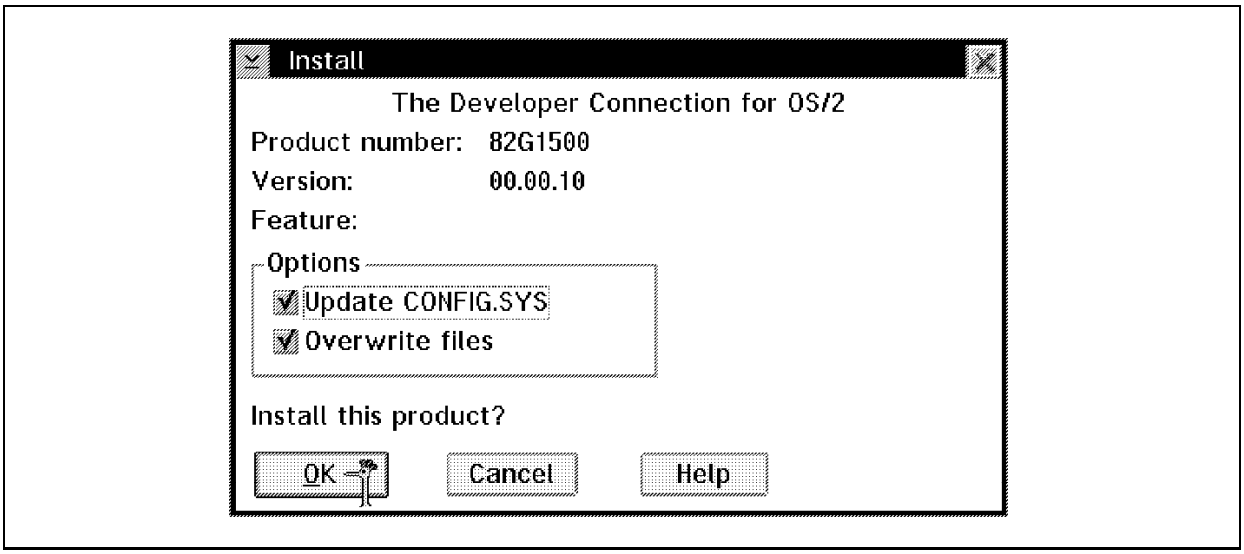


Figure 5. The Developer Connection for OS/2 Volume 10 Install Screen

7. The default options of Update CONFIG.SYS and Overwrite files are recommended and are shown as selected in Figure 5. Select **OK** to proceed with the installation.
8. You have the following options, as shown in Figure 6 on page 20.
   - CATALOG (Run from CD-ROM)
   - BROWSER (Run from CD-ROM)
   - CATALOG (Run from the hard disk)
   - BROWSER (Run from the hard disk)

Option	Space Required
Run from CD-ROM	100KB The catalog and browser programs will be copied to your hard drive, but the catalog data will not. Every time you access the catalog, you will need to place a CD from The Developer Connection for OS/2 Volume 10 into your CD-ROM drive.
Run from the hard disk	10MB The catalog and browser programs and the catalog data will be copied to your hard disk. When you access the catalogs you will use the copy on your hard drive; you will only need The Developer Connection for OS/2 Volume 10 CDs to install products.

The default is **Run from CD-ROM**, and for most users the savings in hard drive space offsets the slower access speed of the CD-ROM drive.

If you want to change to **Run from the hard disk**, unselect the **Run from CD-ROM** and select **Run from the hard disk** for both the catalog and the browser.

On the **Install-directories** dialog you can also specify the directory where you want the catalogs and The Developer Connection for OS/2 program installed. The default directory is `C:\DEVCON`, as shown in Figure 6 on page 20. You can change to a different drive and directory by typing over the default value or by using the **Disk Space...** pushbutton to display a list of the free space available on your hard drives.

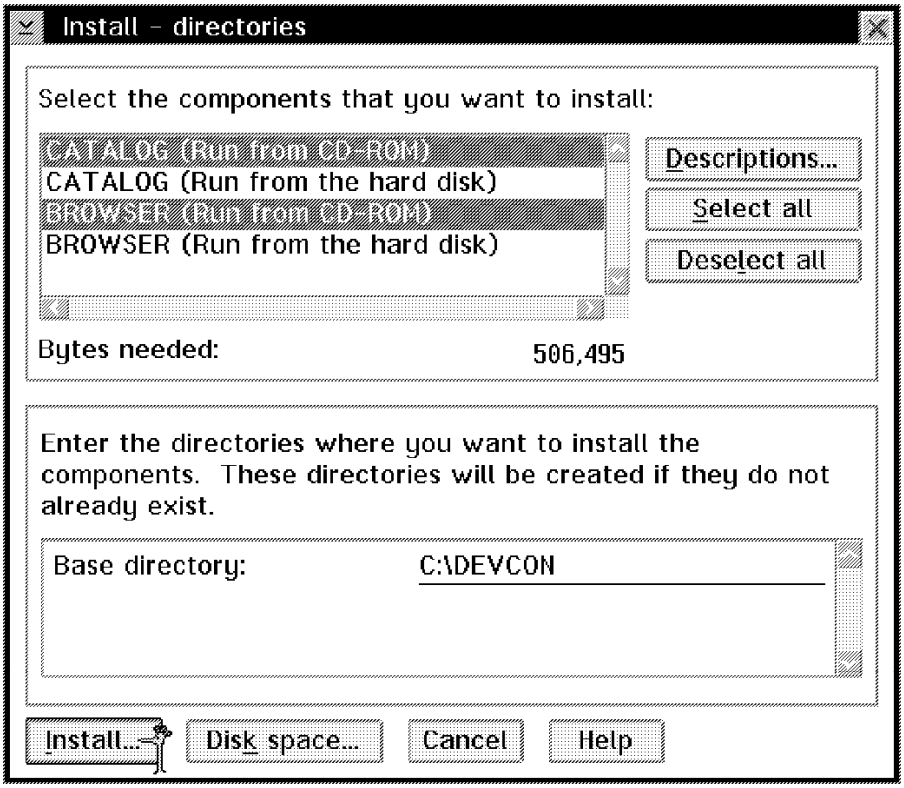


Figure 6. The Developer Connection for OS/2 Volume 10 Install - Directories

9. After you have made your selections on the Install-directories screen shown in Figure 6, you can proceed with the installation by selecting the **Install...** pushbutton.

   The Install progress dialog will display the status of the installation activity, as shown in Figure 7 on page 21.

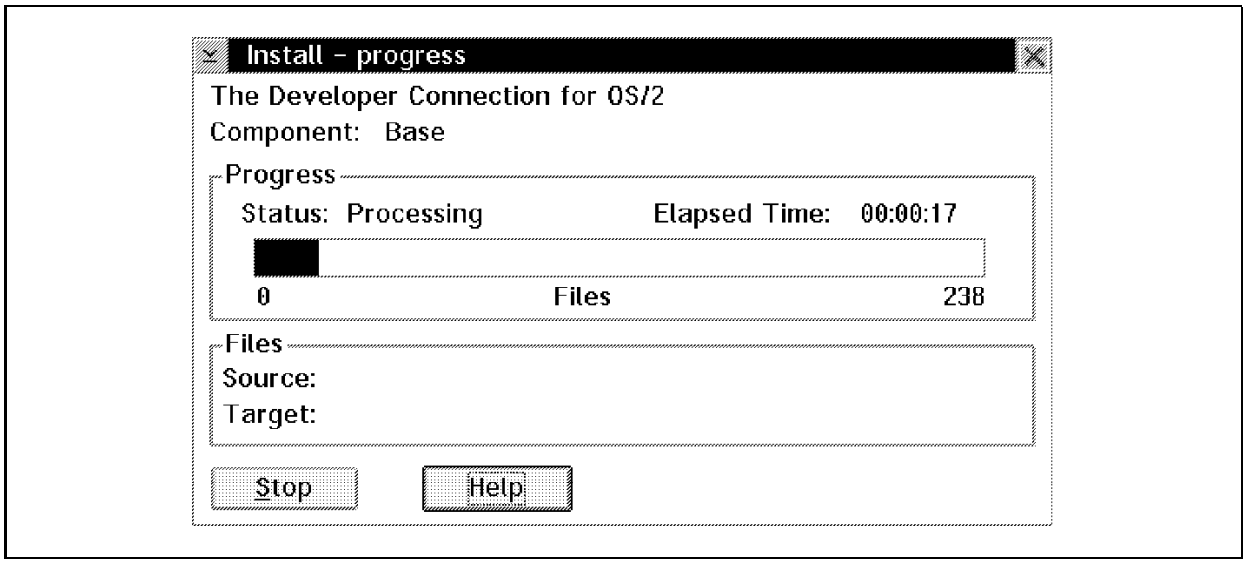


Figure 7. The Developer Connection for OS/2 Volume 10 Install Progress

After all files are copied, the Installation and Maintenance screen will appear, as shown in Figure 8.

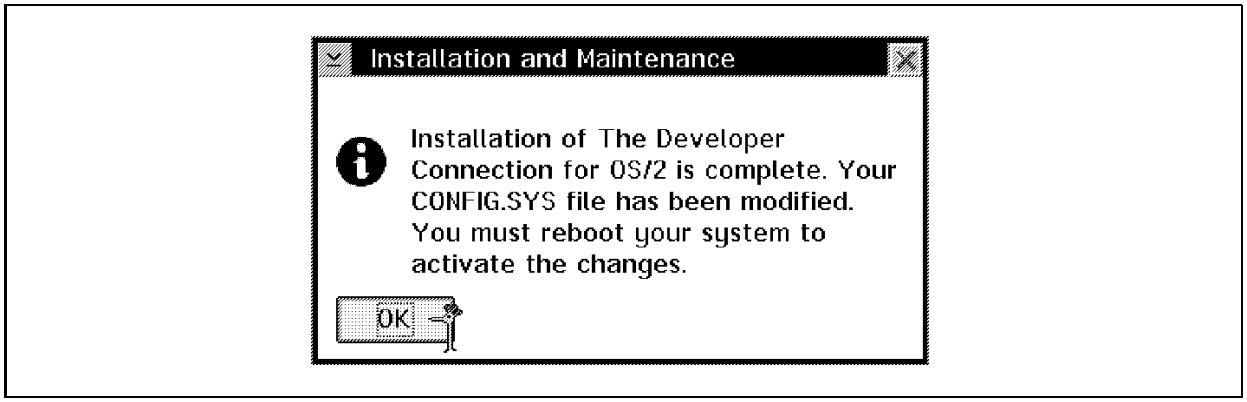


Figure 8. Installation and Maintenance

10. Select **OK** on the Installation and Maintenance screen of Figure 8 and the installation dialogs will close, returning you to The Developer Connection for OS/2 window as shown in Figure 9 on page 22.
11. Select **Exit** on the screen of Figure 9 on page 22 to close the installation window of The Developer Connection for OS/2 Volume 10.
12. Shutdown and reboot your computer before starting The Developer Connection for OS/2.

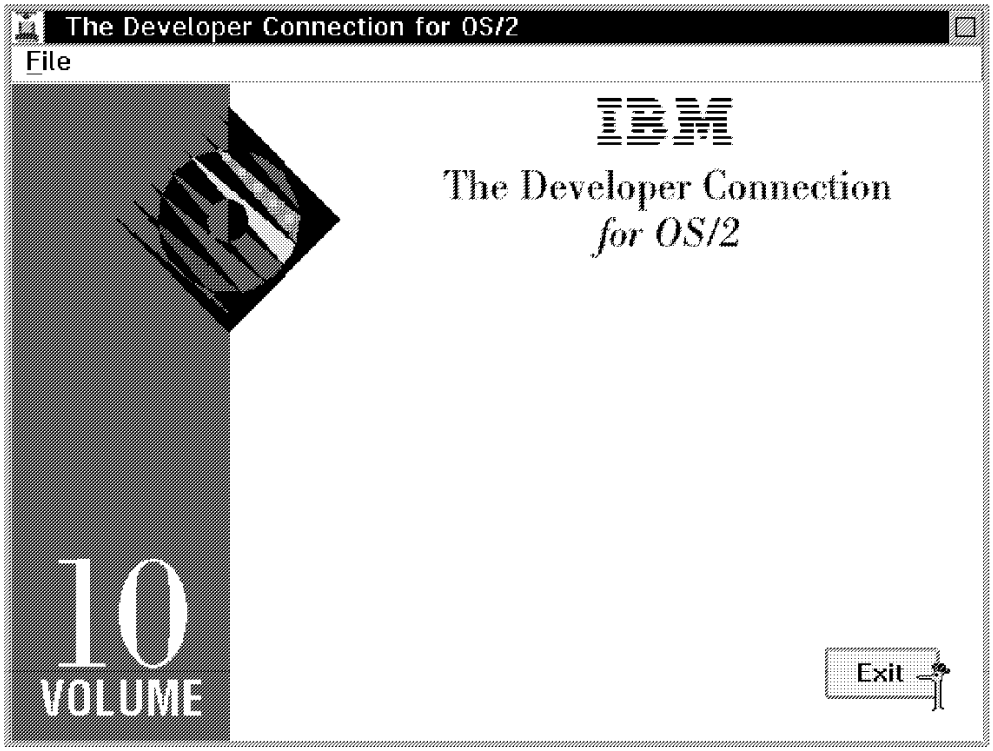


Figure 9. The Developer Connection Installation Finished

2.1.2 Starting The Developer Connection for OS/2 Volume 10

After installing The Developer Connection for OS/2, you will find The Developer Connection folder icon on your desktop, as shown in Figure 10. To start the The Developer Connection for OS/2 Volume 10 catalog, follow these steps:

1. Double click on **The Developer Connection** folder icon on your desktop, as shown in Figure 10. This will open the folder and display the contents, as shown in Figure 11 on page 23.

Figure 10. The Developer Connection Folder on the Desktop

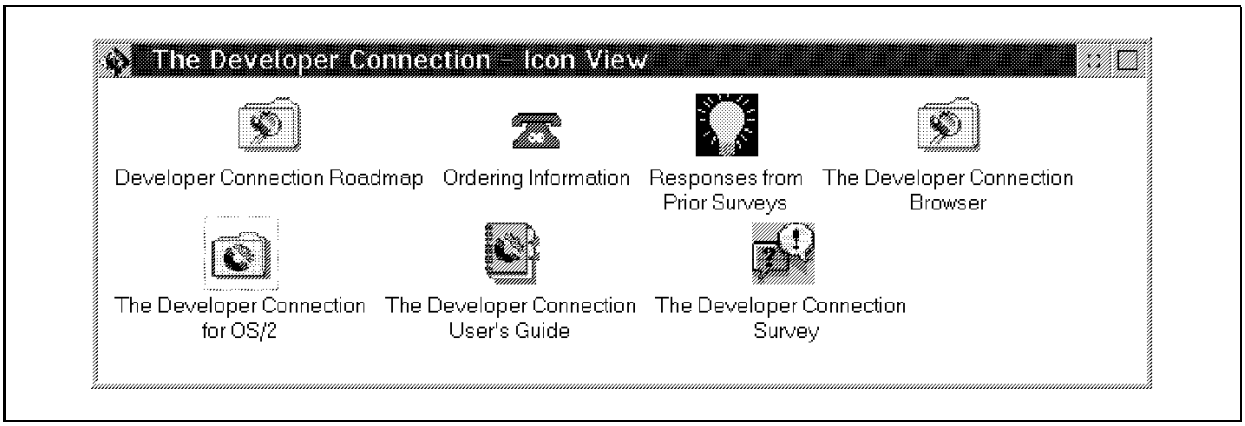


Figure 11. The Developer Connection Folder

2. In the Developer Connection folder, double click on the **The Developer Connection for OS/2** folder to open it, which will give you the **The Developer Connection for OS/2 - Icon View** as shown in Figure 12.

Figure 12. The Developer Connection for OS/2 Folder

3. Double click on **The Developer Connection for OS/2 Catalog** to start the catalog program, which will display to you the Developer Connection Catalog as shown in Figure 25 on page 32. If you selected to run The Developer Connection for OS/2 from CD-ROM, you will be prompted to place The Developer Connection for OS/2 CD into your CD-ROM if you have not already done so.

Note

The first time you run the catalog, the What's New in Volume 10 window will display, as shown in Figure 13 on page 24. You can close it by selecting **Close** from the system menu in the upper-left corner of the window.

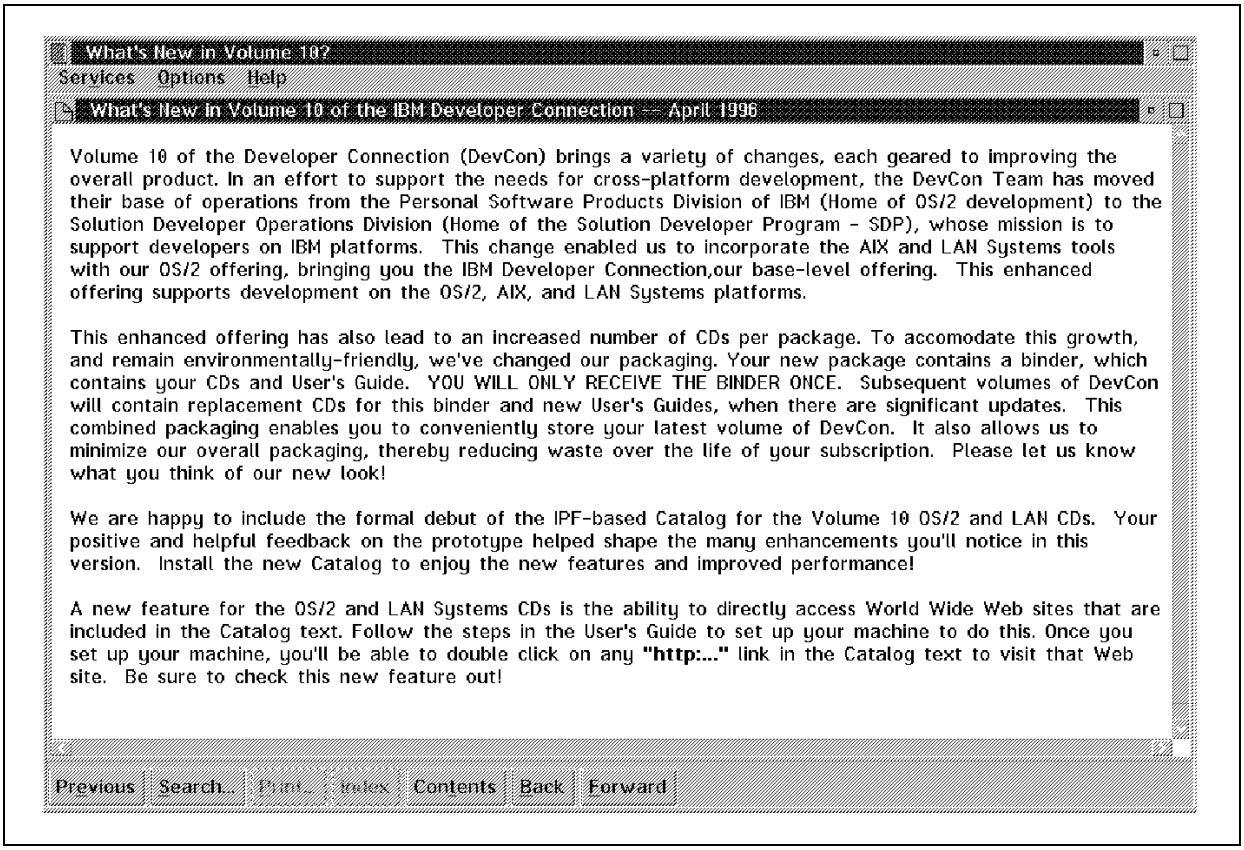


Figure 13. What's New in The Developer Connection for OS/2 Volume 10

2.2 FixPak 17 (XR_W017)

To develop or execute Developer API Extensions applications on OS/2 Warp Version 3 or OS/2 Warp Connect Version 3, you must first install FixPak 17 (XR_W017). It contains a series of changes required by Open32. For versions of OS/2 beyond 3.00, this is not necessary. For example, OS/2 Warp Version 4 does not require the installation of FixPak 17.

To see if you need to install FixPak 17 on your OS/2 Warp system, use the VER /R command, as shown in Figure 14 on page 25. If the Revision is 8.241 or greater, you do not need FixPak 17.

```
[C:\]VER /R

The Operating System/2 Version is 3.00
Revision 8.241

[C:\]
```

Figure 14. Revision Level

2.2.1 Installing FixPak 17

To install FixPak 17:

1. Insert Disc 4 of The Developer Connection for OS/2 Volume 10 in your CD-ROM.
2. Open an OS/2 Windowed Command prompt from the Command Prompt folder.
3. Change to your CD-ROM drive, as shown in the first line of Figure 15.

```
[C: ]E:

[E:\]CD \SERVICES\FIXPAK

[E:\]SERVICE
```

Figure 15. Starting Corrective Service Facility

4. Change to the SERVICES FIXPAK directory, as shown on the second line of Figure 15.
5. Enter SERVICE to begin Corrective Service Facility, as shown on the third line of Figure 15.
6. The Corrective Service Facility Product Information dialog box will display, as shown in Figure 16 on page 26. Select **OK** to continue with the installation of FixPak 17.

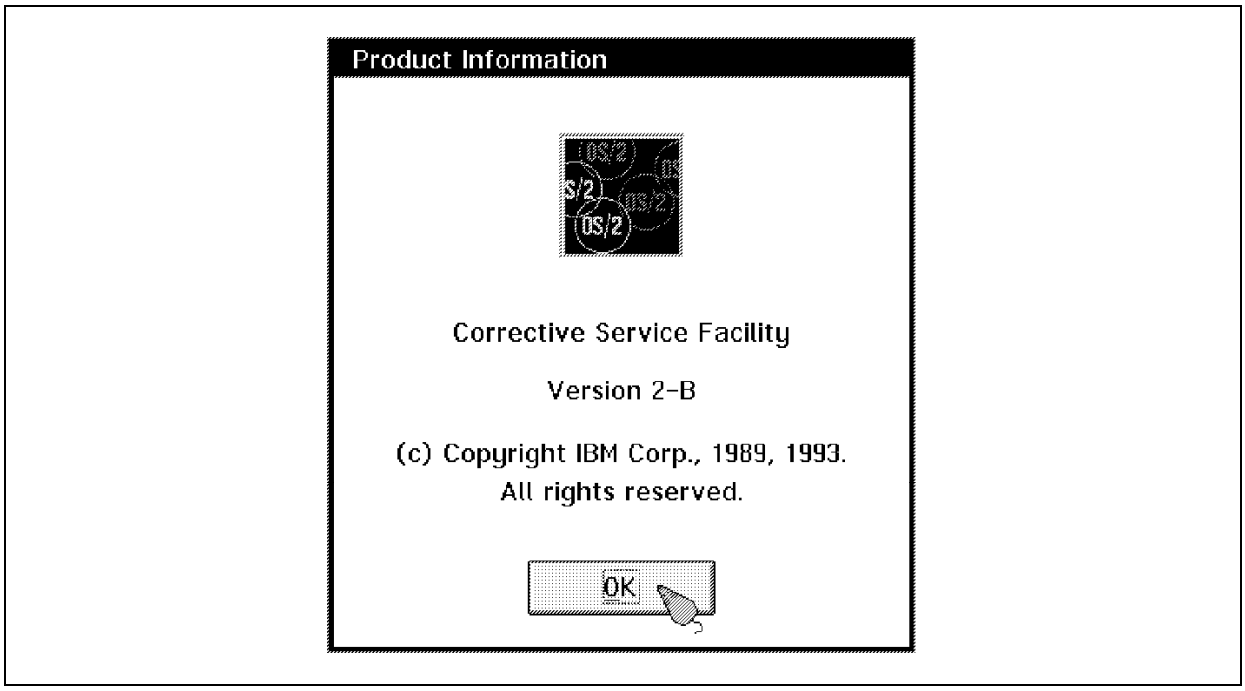


Figure 16. Corrective Service Facility Product Information

7. You will then see the Select Source Drive dialog box, shown in Figure 17. Select your CD-ROM drive and press **OK** to continue.

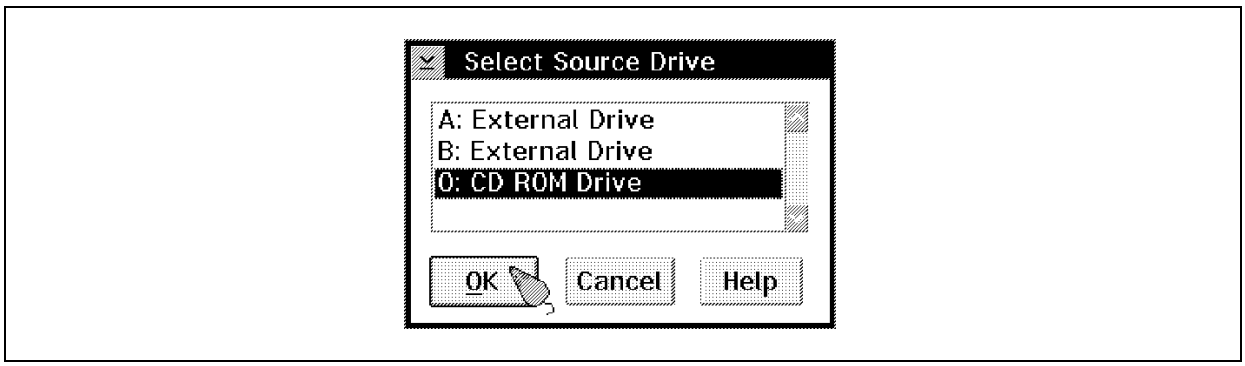


Figure 17. Select Source Drive Dialog Box

You will see the wait box shown in Figure 18 on page 27. Please be patient while the Corrective Service Facility inspects your system.

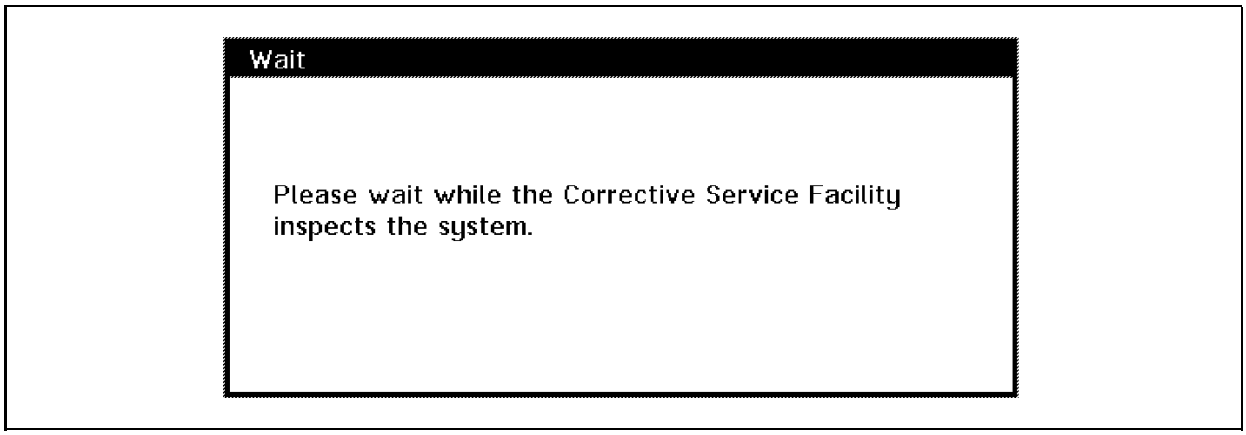


Figure 18. Please be Patient

8. After the inspection is completed, the Corrective Service Facility will display a list of serviceable products. In most installations, it will list one IBM OS/2 Base Operating System and, if you installed multimedia support, one IBM Multimedia Presentation Manager/2, as shown in Figure 19. If you have more than one copy of OS/2, deselect any copies you do not want serviced.

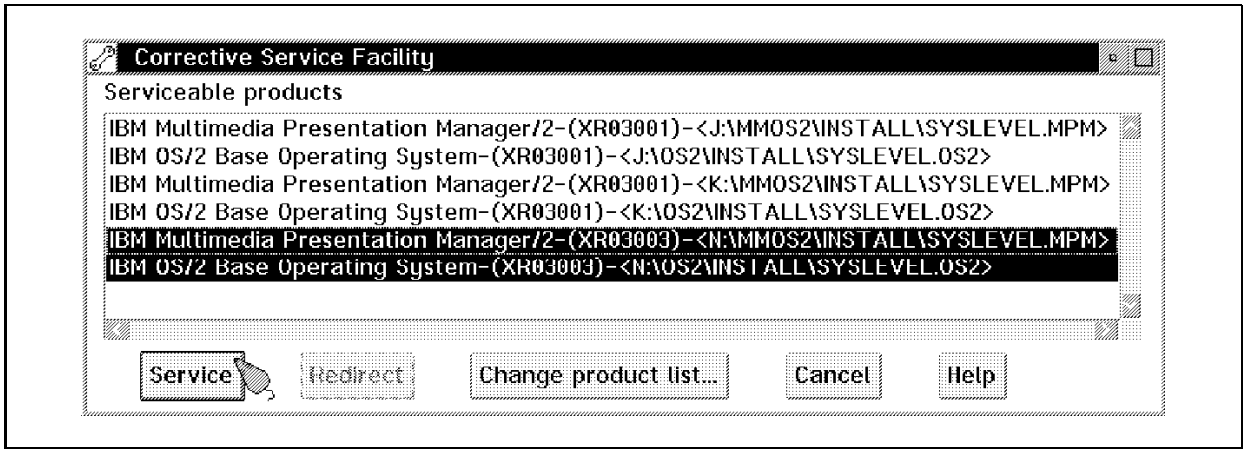


Figure 19. Corrective Service Facility: Serviceable Products

9. After you have selected which products you want serviced, select the **Service** button to begin updating the components.
10. The Corrective Service Facility will prompt you for an Archive path, as shown in Figure 20 on page 28. The path must be unique for each product to be serviced. When you have typed in a path for each product, select the **OK** button.

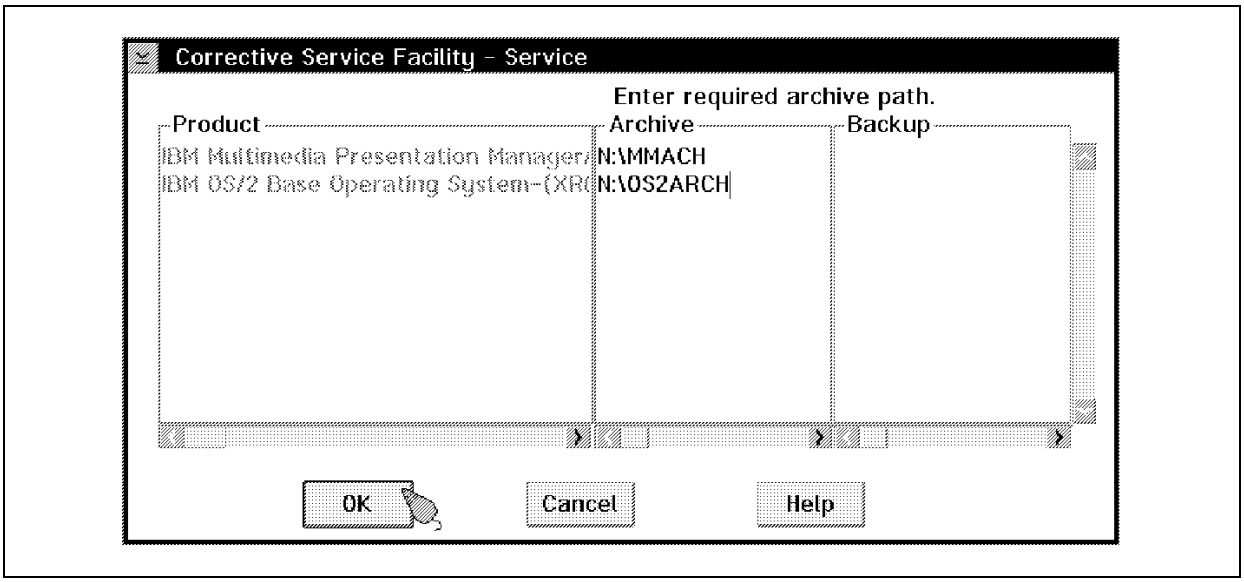


Figure 20. Corrective Service Facility: Archive Path Prompt

11. The Corrective Service Facility needs to update files which are in use by OS/2, so it will show you a list of files which are locked, as shown in Figure 21. Select **Continue** to allow it to handle the locked files automatically.

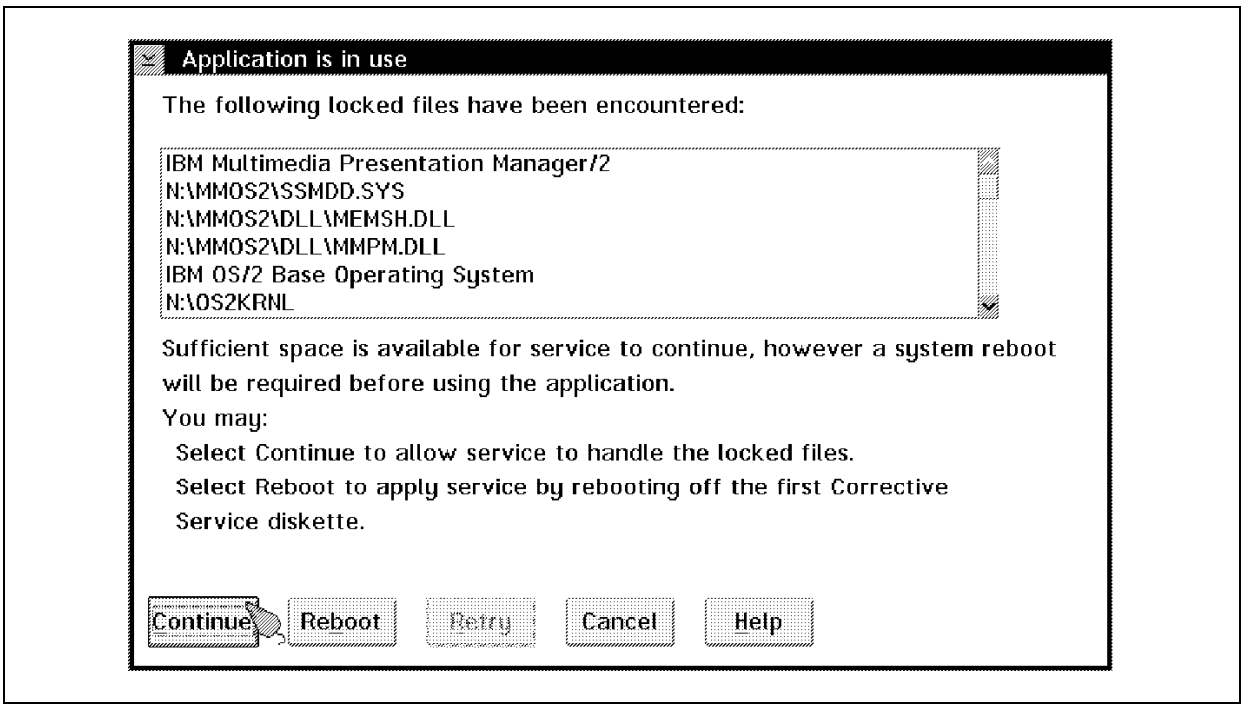


Figure 21. Corrective Service Facility: Locked Files

The Corrective Service Facility will display a progress window, shown in Figure 22 on page 29, while it updates files. This process can take up to forty minutes, so be patient. You may not use OS/2 while the system is being serviced.

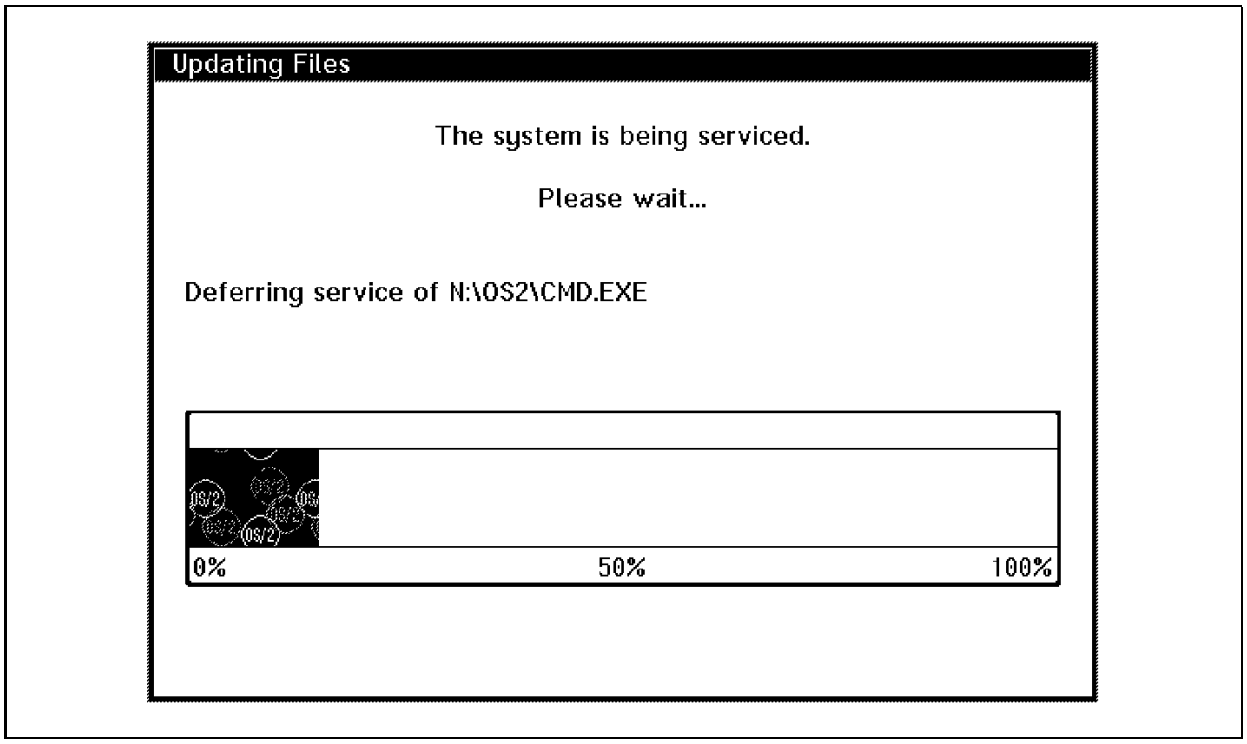


Figure 22. Corrective Service Facility: Progress Window

12. During the service process, you may be prompted for Service Permission, as shown in Figure 23 on page 30. You should select **OK** to allow Corrective Service Facility to update the file.

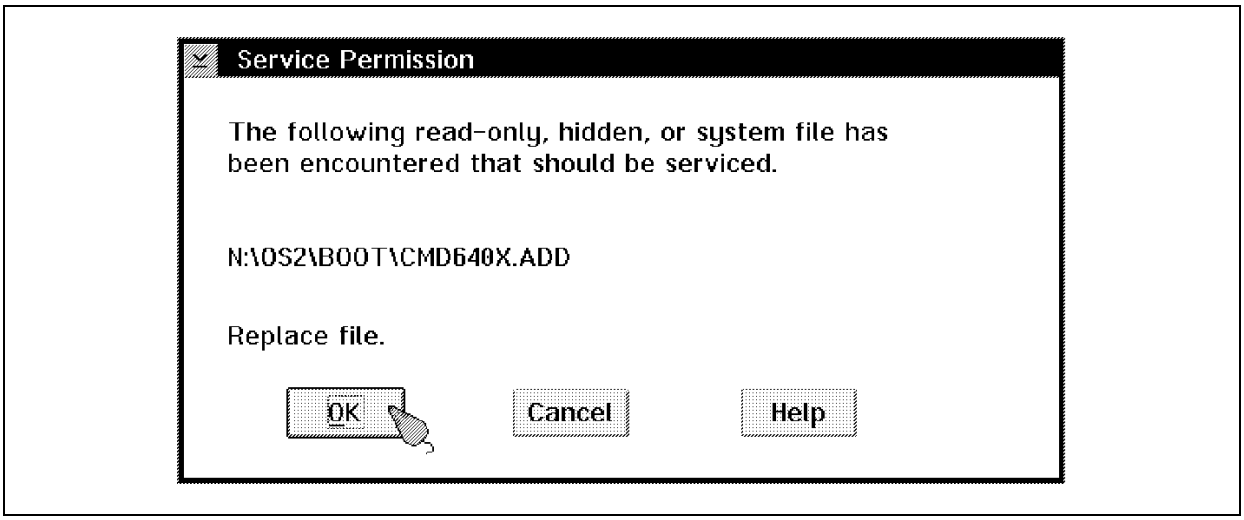


Figure 23. Corrective Service Facility: Service Permission

13. When Corrective Service Facility finishes updating your system, the Corrective Service Facility Message box will display, as shown in Figure 24. Select **Exit** to close Corrective Service Facility.

Figure 24. Corrective Service Facility: Service Complete

14. Shut down your computer and reboot. During OS/2 start up, locked files which could not be accessed while OS/2 was running will be updated. Your computer may automatically reboot after locked files are updated.

2.3 OS/2 Warp Toolkit

One of the tools that you will need for developing applications that utilize Developer API Extensions for OS/2 Warp is the OS/2 Warp Toolkit.

This section briefly describes installing the OS/2 Warp Toolkit from The Developer Connection for OS/2 Volume 10 CD-ROM.

2.3.1 Installing OS/2 Warp Toolkit

The Developer Connection for OS/2 Volume 10 includes the latest version of the Toolkit package. This is the toolkit you will need to use to develop Open32 applications on OS/2 Warp.

You can install the OS/2 Warp Toolkit from the The Developer Connection for OS/2 catalog by performing the following steps:

1. Start The Developer Connection for OS/2 Catalog by following the steps outlined in 2.1.2, “Starting The Developer Connection for OS/2 Volume 10” on page 22. The catalog of The Developer Connection for OS/2 Products will then be displayed, as shown in Figure 25 on page 32.

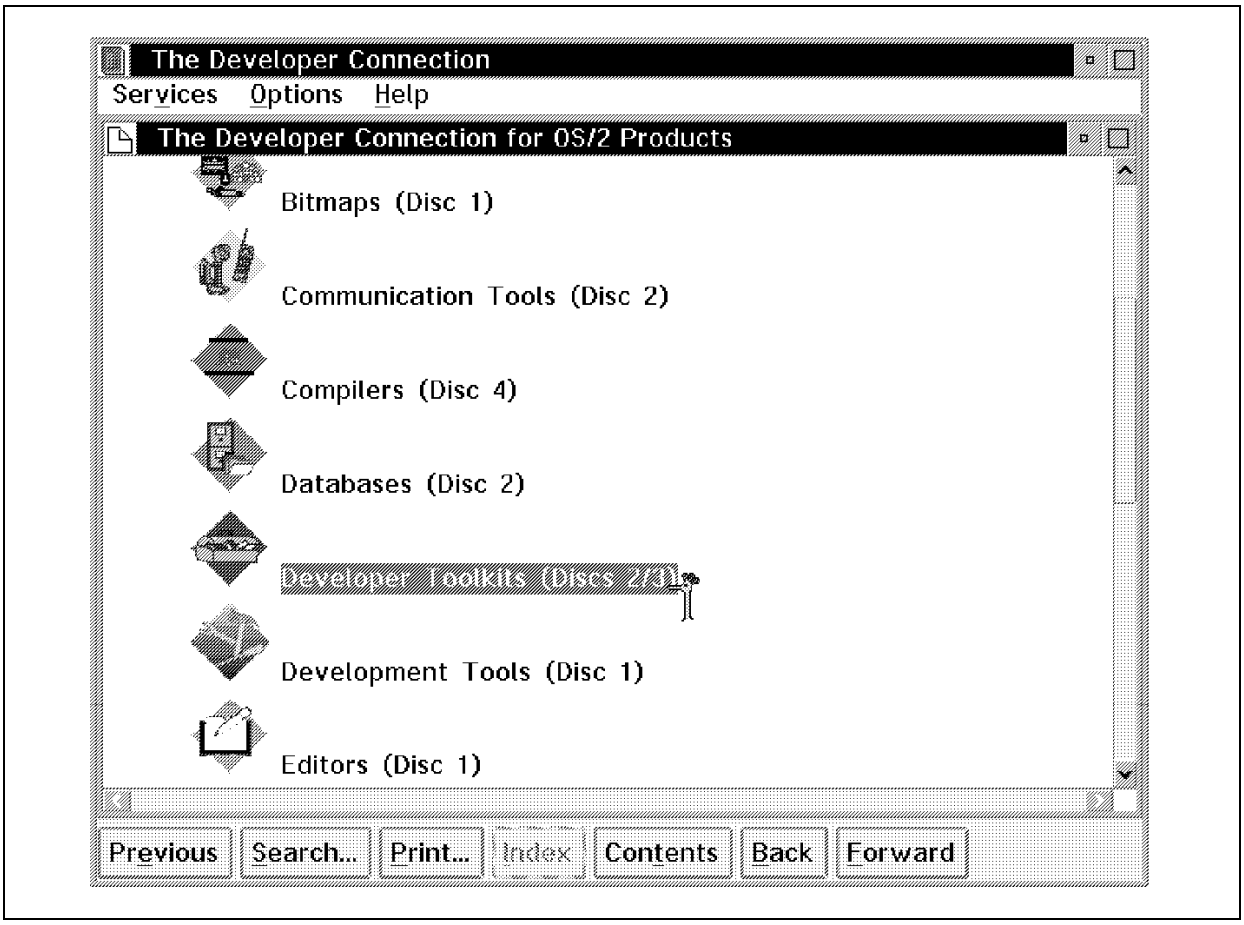


Figure 25. The Developer Connection for OS/2 Volume 10 Catalog

2. Double click on the item **Developer Toolkits (Discs 2/3)**, shown highlighted in Figure 25. The menu shown in Figure 26 on page 33 will be displayed.
3. Double click on **Toolkits**.

Figure 26. Developer Toolkits Menu

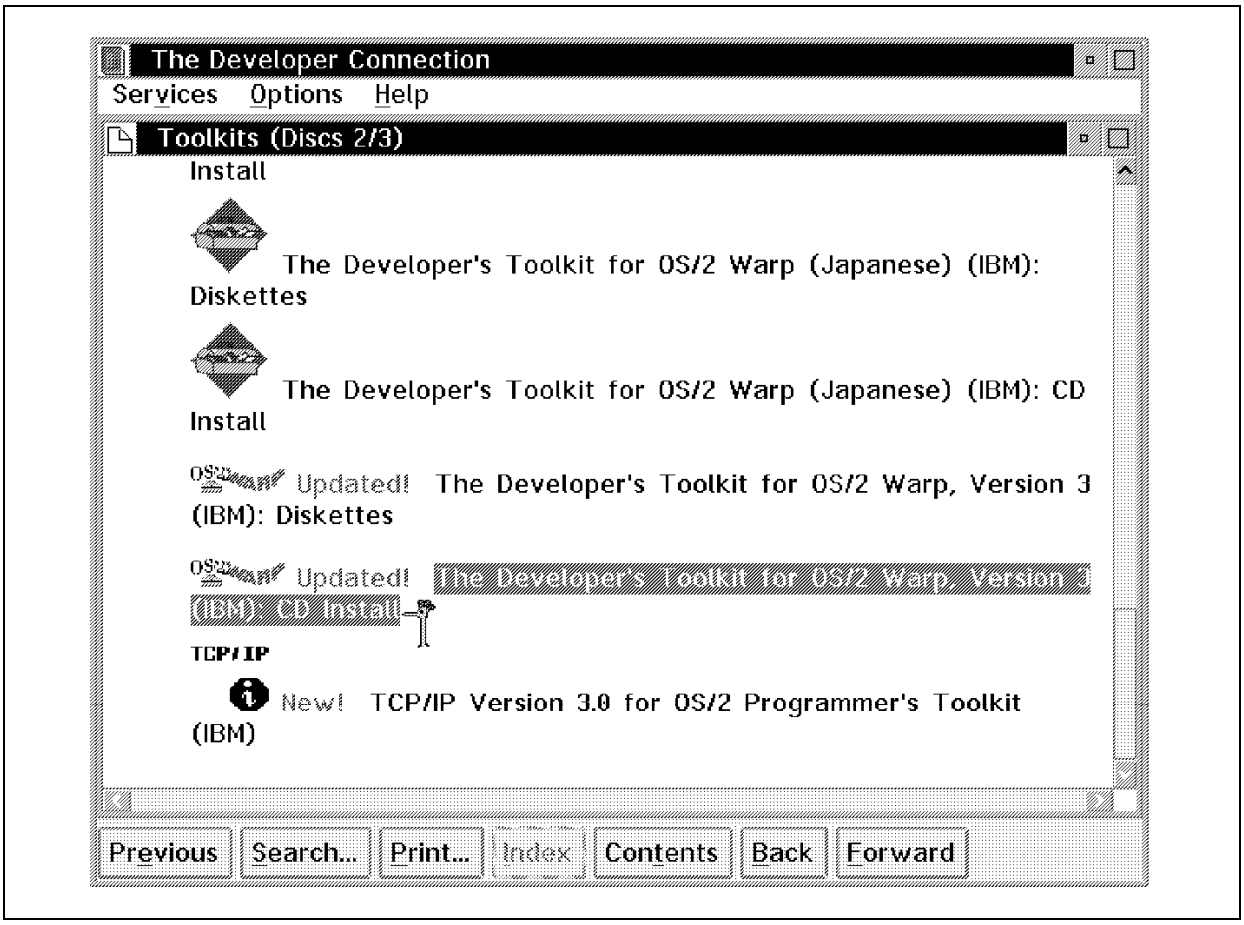


Figure 27. Toolkits Available in The Developer Connection for OS/2

4. You will see the list of toolkits available on The Developer Connection for OS/2. At the bottom of the list is **The Developer's Toolkit for OS/2 Warp, Version 3 (IBM): CD Install**, as shown in Figure 27. Double click on it to see information about the OS/2 Warp Toolkit, as shown in Figure 28 on page 35.

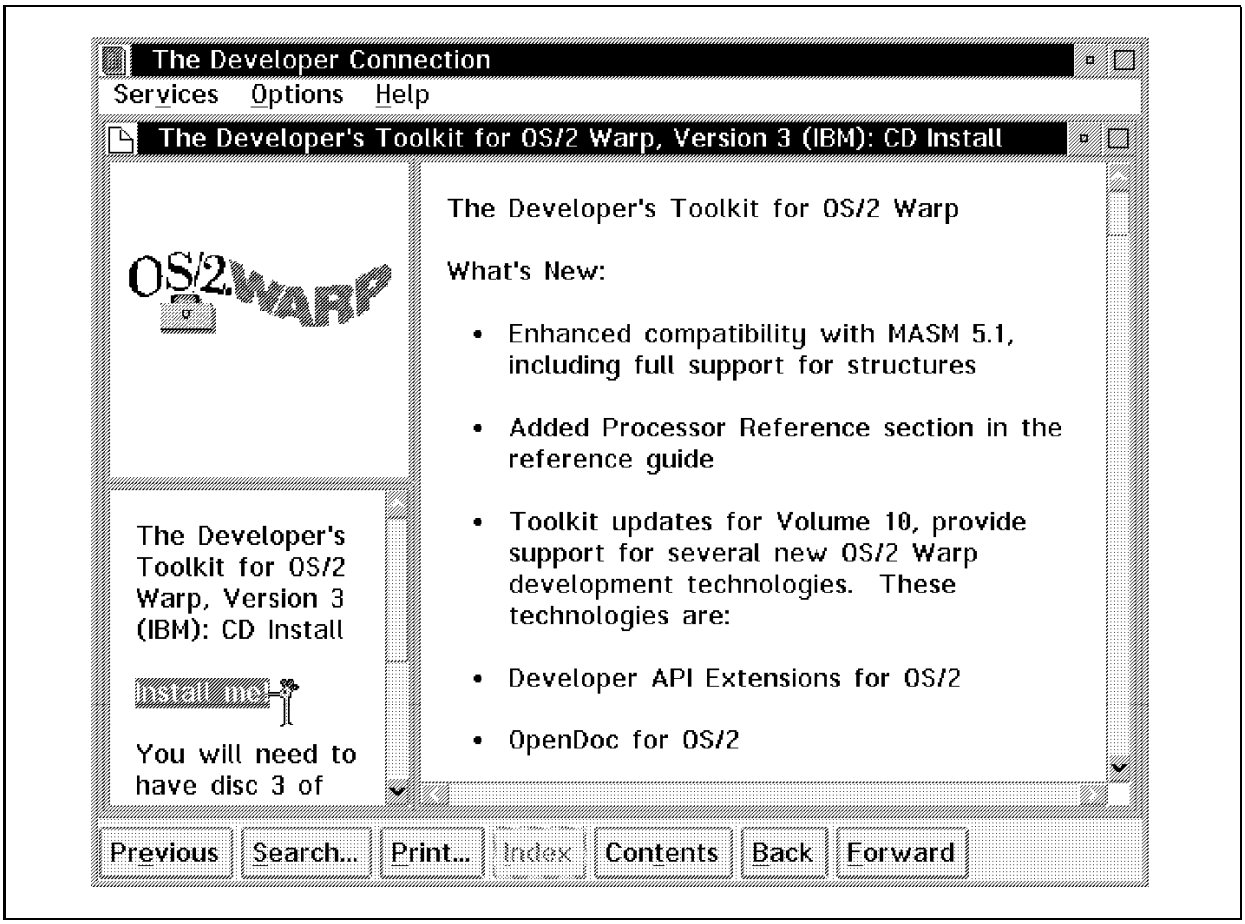


Figure 28. Information about OS/2 Warp Toolkit

5. Double click on **Install me!** to start the OS/2 Warp Toolkit installation.

   If the Disc request dialog, shown in Figure 29 on page 36, appears, it could be because the disc in the CD-ROM drive is not Disc 3 or the drive letter is not a valid CD-ROM drive. Make sure that Disc 3 and the drive letter are correct. Select the **OK** pushbutton of the Disc request dialog to continue the installation.

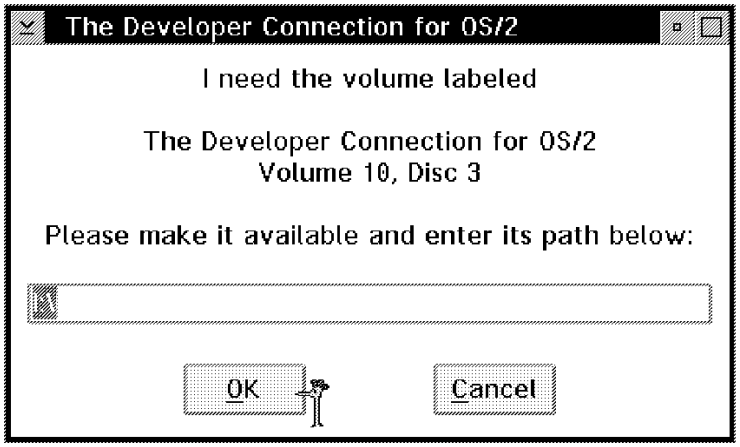


Figure 29. Disc Request Screen

The IBM Developer's Toolkit for OS/2 Warp Installation Warning window as shown in Figure 30 will be displayed before OS/2 Warp Toolkit installation begins. If you have not installed FixPak 17, see 2.2, “FixPak 17 (XR_W017)” on page 24.

Note

You will receive the warning even after FixPak 17 is installed. If you are not sure if you need to install the FixPak 17, or if you do not know if it has already been installed, see 2.2, “FixPak 17 (XR_W017)” on page 24.

6. When you are ready to continue with the installation of the OS/2 Warp Toolkit, push **OK** on the warning dialog.

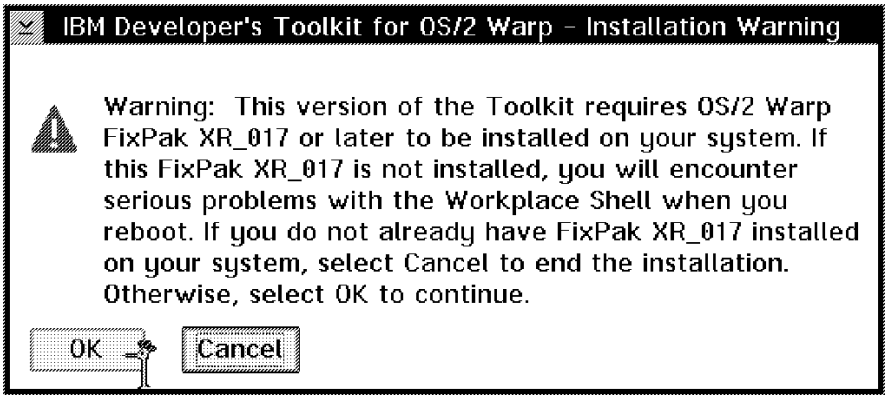


Figure 30. Installation-Warning Screen

The Developer's Toolkit is comprised of several items. You can select to install all or part of the items from the list of installable components shown in Figure 31 on page 37.

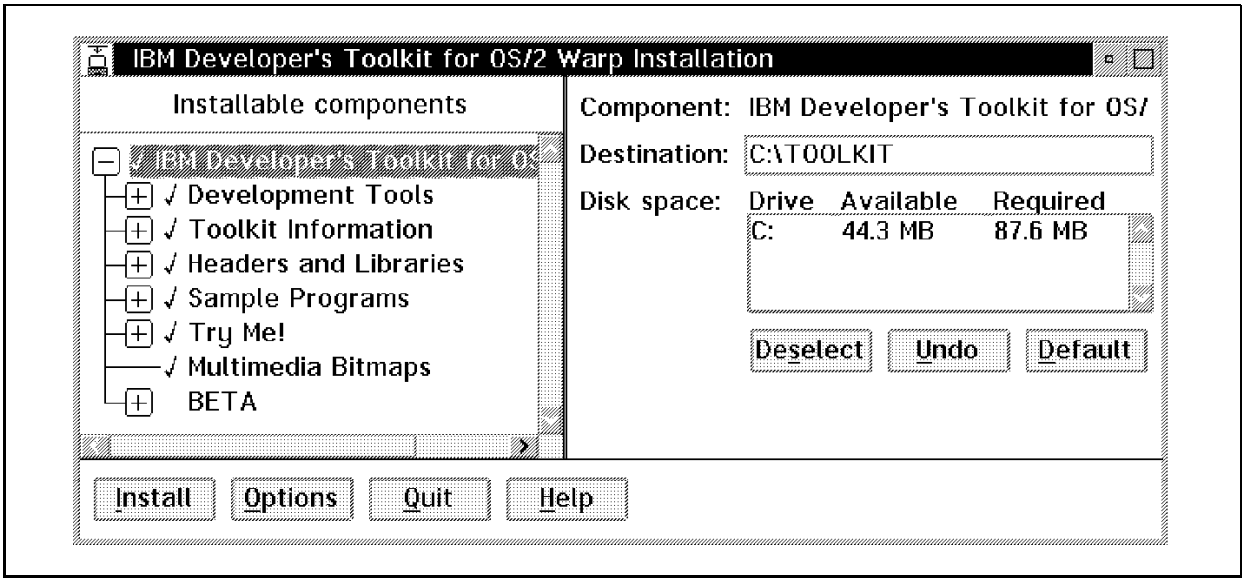


Figure 31. Installation Selection Screen

Following is an expanded list of items that can be installed from the installable component list of the IBM Developer's Toolkit for OS/2 Warp. The items that are required or recommended to be installed to support Open32 application development are noted below. If you do not want to install all of the components you can select only the items marked as required.

- Development Tools
  - Base Tools (Required)
- Toolkit Information
  - Control Program Guide and Reference
  - Graphics Program Guide and Reference
  - Presentation Manager Guide and Reference
  - IBM Developer API Extensions for OS/2 Guide (Recommended)
- Headers and Libraries
  - OS/2,PM & MM C/C++ Headers (Required)
  - Libraries (Required)
- Sample Programs

    - Developer API Extensions Samples (Recommended)
- Try Me!
    - Development Tools
    - OpenDoc Tools
    - Sample Programs
- Multimedia Bitmaps
- BETA
    - BETA Entertainment Support

If you choose to install all of the components for the Developer's Toolkit for OS/2 Warp you must specify a hard drive with over 94MB of free space. By changing the destination drive the available space on that drive will be displayed in the Disk space information area.

If you select to install only the required and recommended items as shown in the above list, you need only 18MB of space on your hard disk as shown in Figure 32.

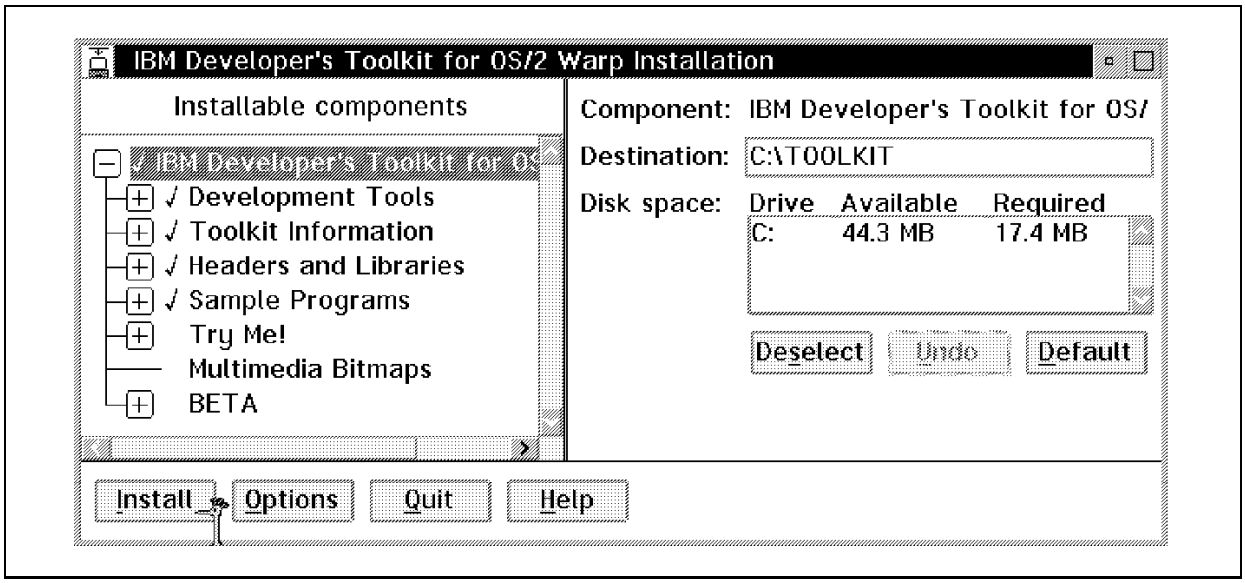


Figure 32. Installation Selection (Minimum) Screen

To select or deselect an item for installation, highlight the item in the list and press the **Select** or **Deselect** pushbutton. Only the items with a check mark next to their names will be installed.

If you need to change the installation options, such as whether or not to update CONFIG.SYS, push the **Options** button at the bottom of the

installation window. This will display the **Installation options** dialog shown in Figure 33 on page 39. For most installations, the default options shown are correct and do not require any changes.

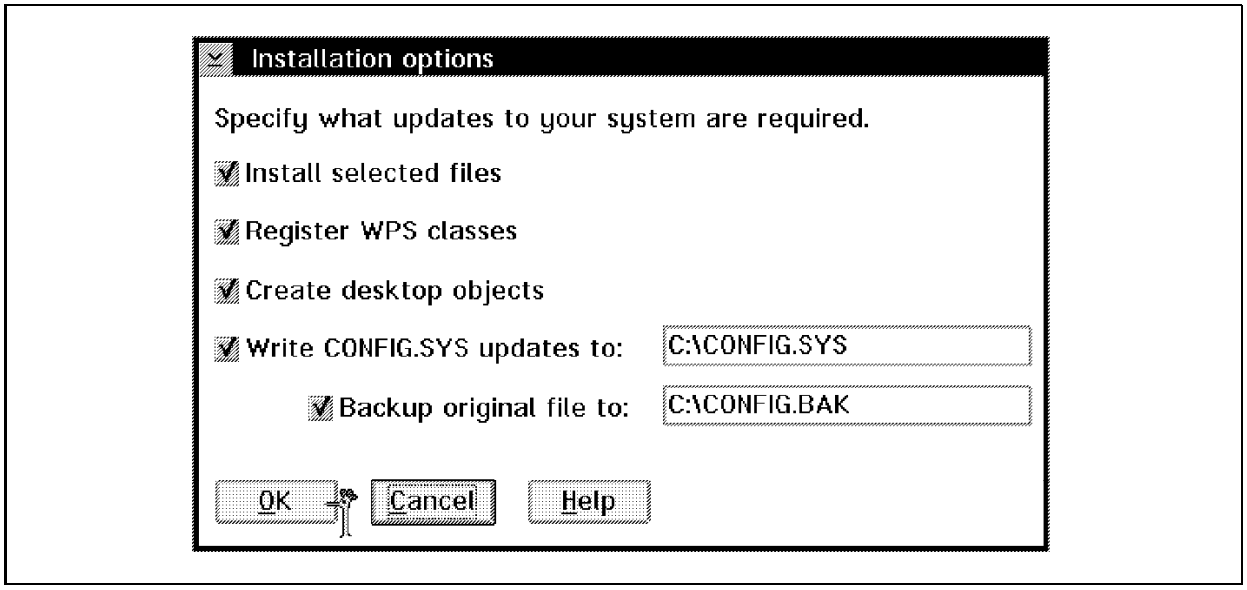


Figure 33. Installation Options Dialog

7. After you have set the options as you want, press the **OK** button to close the dialog and save your option settings.
8. Select the **Install** button on the installation screen shown in Figure 32 on page 38 to start the install of the IBM Developer's Toolkit for OS/2 Warp.
9. The install program will begin to copy files to your hard drive. The **Installation Status** window will appear as shown in Figure 34 on page 40 to show the progress of the installation.

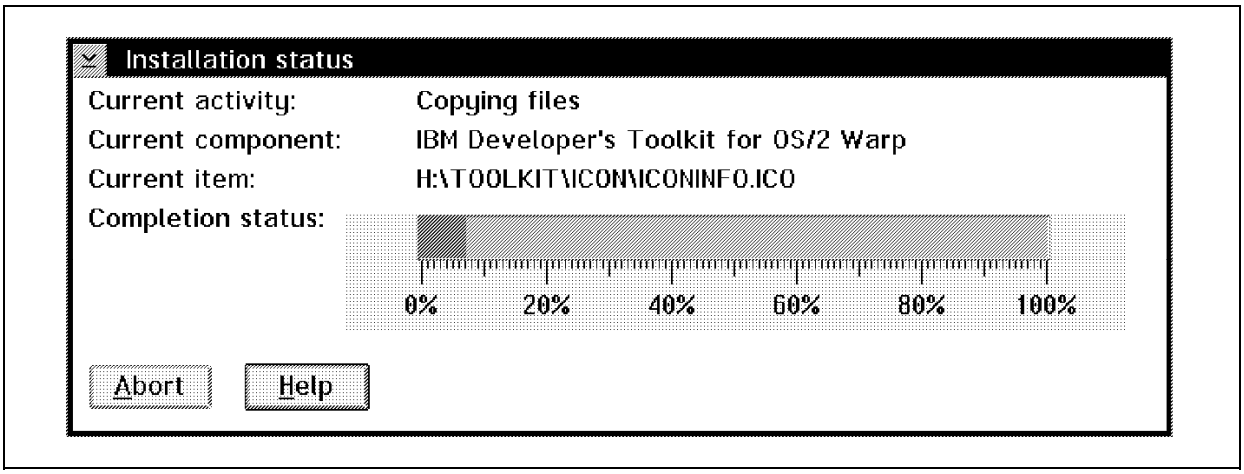


Figure 34. Installation Status Screen

10. Once all the files have been copied you will be informed of the successful installation with the Installation Status window as shown in Figure 35.

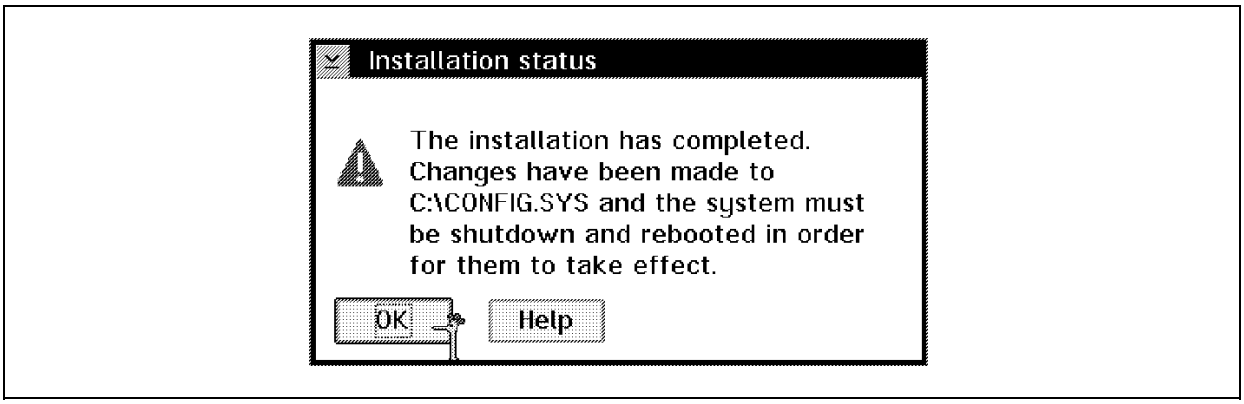


Figure 35. Installation Status Screen

11. Select **OK** to close the installer.

    After the installation has completed, you will need to shutdown and reboot your computer, because CONFIG.SYS has been changed by the installation of the IBM Developer's Toolkit for OS/2 Warp. Rebooting the computer will make these changes take effect and complete the installation process.

2.3.2 Configuring the Resource Compiler

The Resource Compiler is used in OS/2 program development to convert human-readable text descriptions of program resources into binary format. The resources for your Open32 application must be compiled with the latest

version of the Resource Compiler, which ships with the OS/2 Warp Toolkit on The Developer Connection for OS/2. This version supports string IDs for application resources, just as Windows does.

Note

You should check your Resource Compiler even if you do not plan to use string IDs. It is important that you always use the latest version of the Resource Compiler for all Developer API Extensions application development.

There are many ways to find and rename the Resource Compilers which will not work with Open32 application development. We will outline one way which uses the OS/2 Workplace Shell.

To make sure that only the latest Resource Compiler is used:

1. Select **Find** from the launch pad as shown in Figure 36.

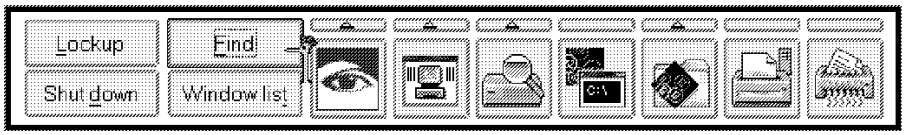


Figure 36. Launch Pad

2. The Find Objects window will be displayed. Enter the name **RC.EXE** as shown in Figure 37, then select the **Find** button.

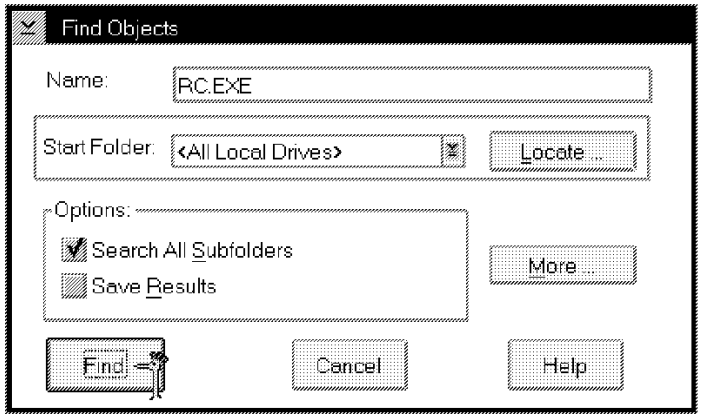


Figure 37. Find Objects

The Searching Progress window will be displayed as shown in Figure 38 on page 42.

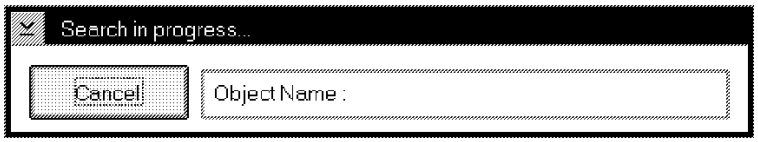


Figure 38. Searching Progress

The find program will now search all of your hard disk drives for the RC.EXE file. The time the search takes depends on the number of directories and files on your hard disk drives. When all the drives have been searched, a Find Results Screen - RC.EXE window similar to the one in Figure 39 will be displayed.

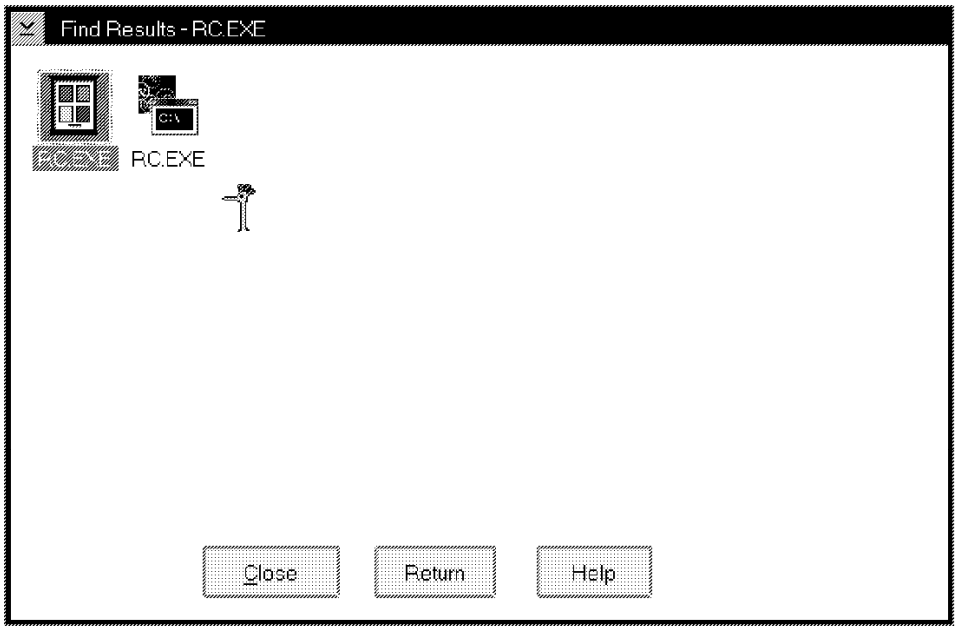


Figure 39. Find Results

3. Click the second mouse button on the first RC.EXE to display the object's menu. Select the **Settings** menu item. The file's Settings notebook will be displayed. Select the **File** tab to show the notebook page in Figure 40 on page 43

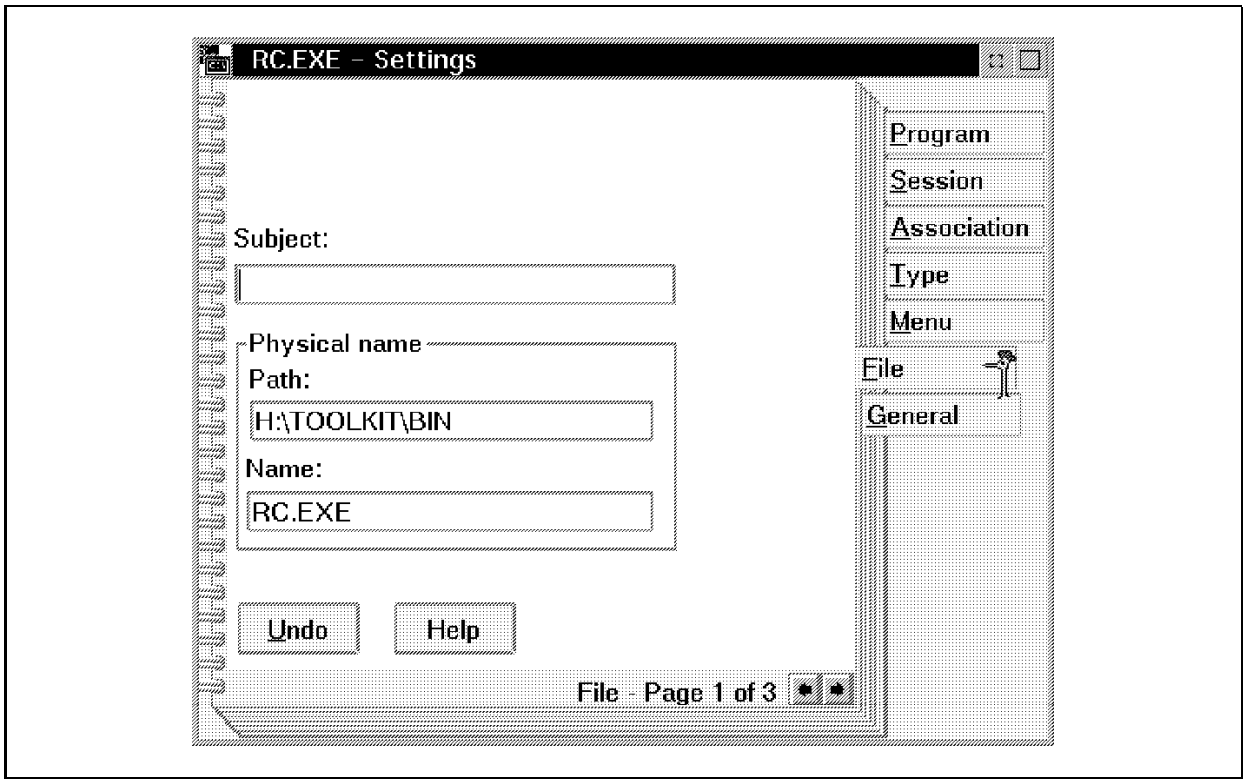


Figure 40. RC.EXE: Settings Notebook--File tab

4. Select the right arrow in the lower right corner of the Settings notebook. This will display a notebook page similar to the one shown in Figure 41 on page 44.

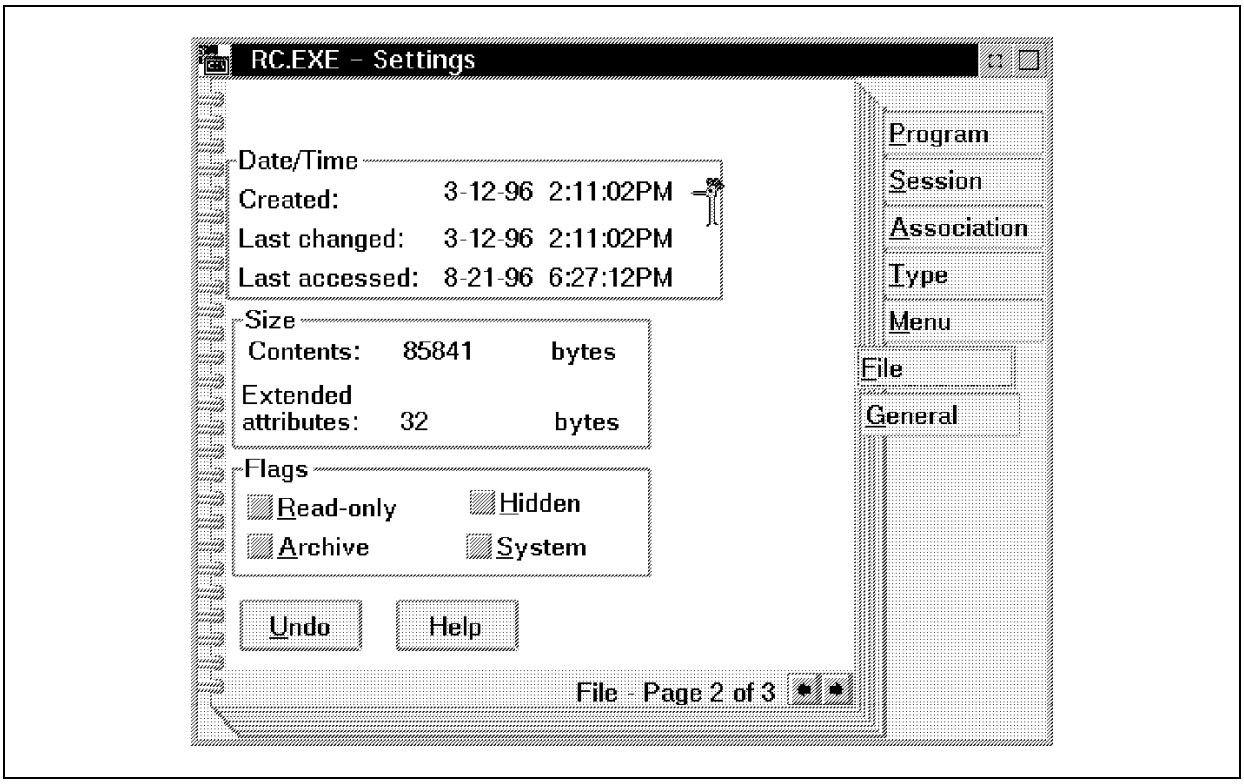


Figure 41. Correct Resource Compiler File

The version of Resource Compiler that comes on The Developer Connection for OS/2 Volume 10 IBM Developer's Toolkit for OS/2 Warp has a creation date of 3-12-96 as shown in Figure 41.

5. For all the Resource Compilers that have creation dates earlier than 3-12-96, you will need to rename them so they are not used in place of the RC.EXE from the IBM Developer's Toolkit for OS/2 Warp.

   For example the Resource Compiler that comes with OS/2 Warp can be seen in Figure 42 on page 45 with a creation date of 10-31-94. Figure 43 on page 46 shows renaming RC.EXE to RC.EXO in the title area of the General tab of the Settings notebook. After typing in the new name in the title area, close the Settings notebook to make the change.

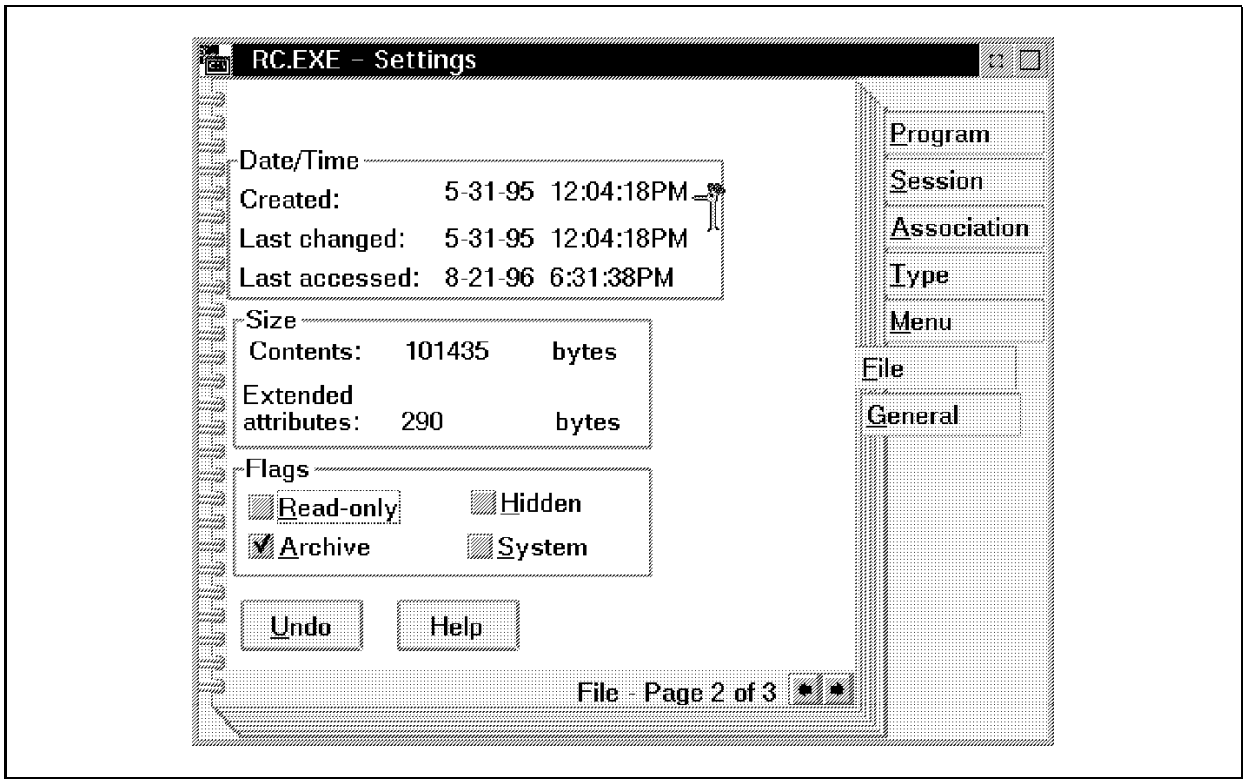


Figure 42. Earlier Resource Compiler File

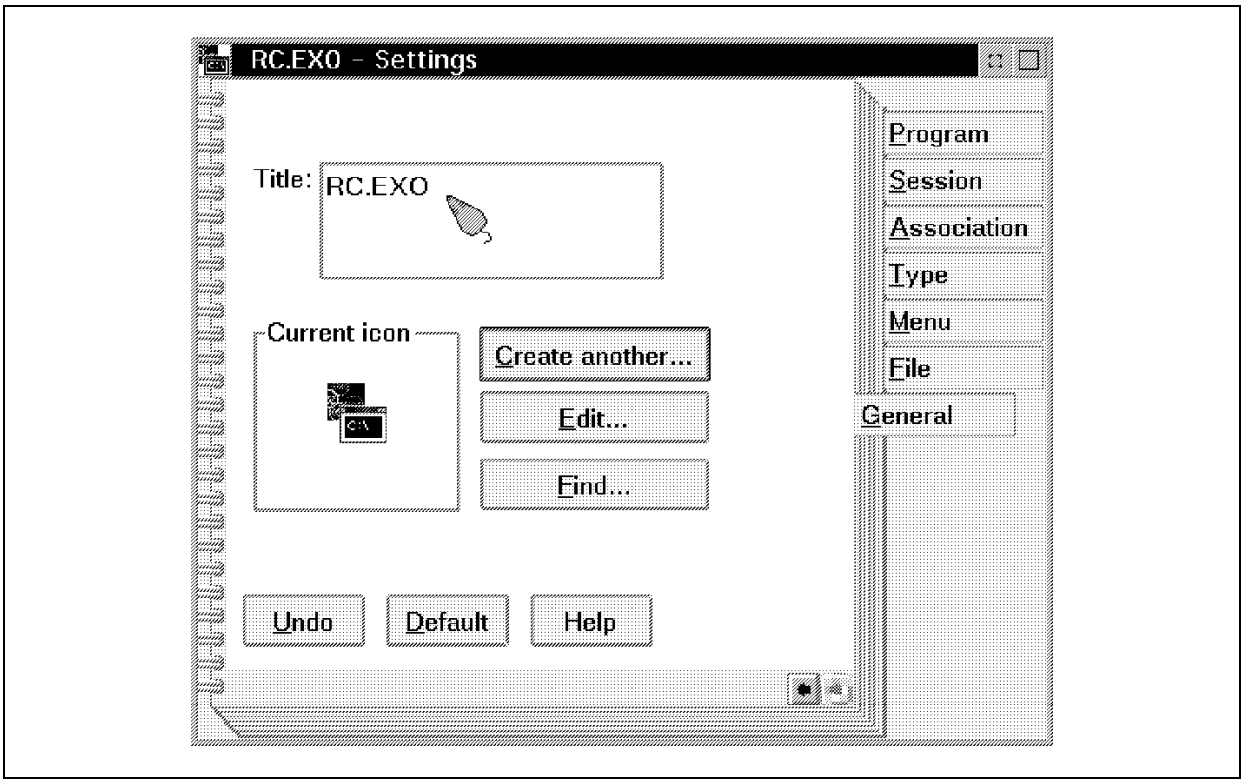


Figure 43. Other Resource Compiler General tab

6. Repeat steps 3 through 5 for every file in the Find Results dialog box. When you are finished, the only file with the name RC.EXE should be the one installed with the OS/2 Warp Toolkit, dated 11-15-95.
7. Close the Find Results - RC.EXE dialog box by selecting the **Close** pushbutton.

2.3.3 Where to Learn More About the Toolkit

Once you have installed the Toolkit, you can find more information about the OS/2 Warp Toolkit in the Toolkit Information folder. Figure 44 on page 47 shows the different on-line documents that are available.

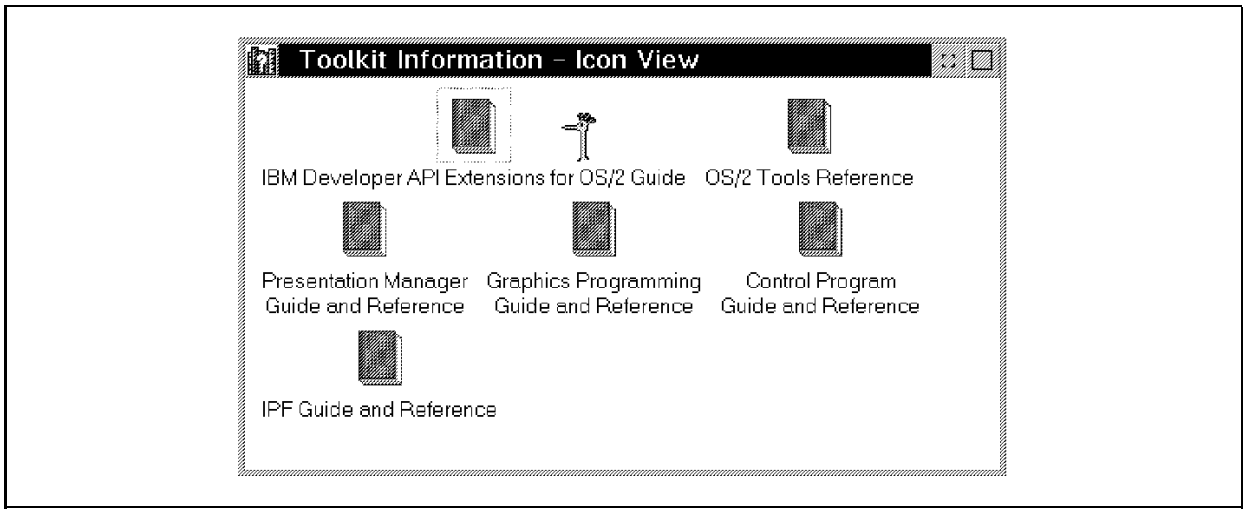


Figure 44. Toolkit Information Folder

Depending on the components you selected to install on the installation menu as shown in Figure 31 on page 37, the documents you have in the Toolkit Information folder may not match those shown in Figure 44.

2.4 SMART

The SMART tool from One Up Corporation automates the conversion of Win32 resources to OS/2. It will translate your Resource Compiler file into OS/2 format and convert Win32 format bitmaps and icons to OS/2 format.

2.4.1 Installing SMART

To install SMART:

1. Start the Developer Connect catalog by following the steps outlined in 2.1.2, “Starting The Developer Connection for OS/2 Volume 10” on page 22.
2. The catalog of The Developer Connection for OS/2 Products will then be displayed, as shown in Figure 45 on page 48.
3. Double click on **Development Tools (Disc 1)** from The Developer Connection for OS/2 Products main menu shown in Figure 45 on page 48.

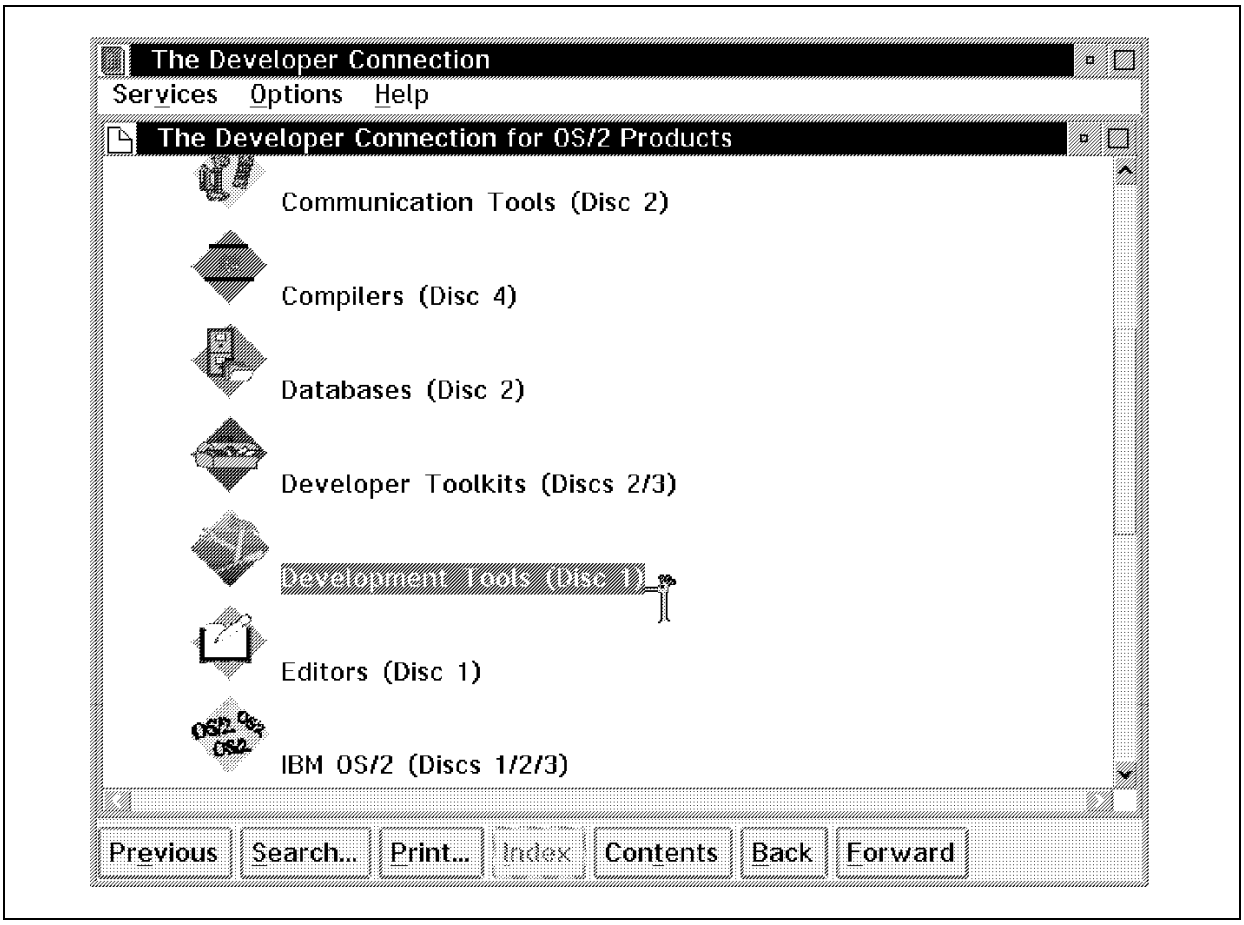


Figure 45. The Developer Connection for OS/2 Products Main Menu

4. Double click on **SMART-Version 2.1 (One Up Corp)**, shown in Figure 46 on page 49.

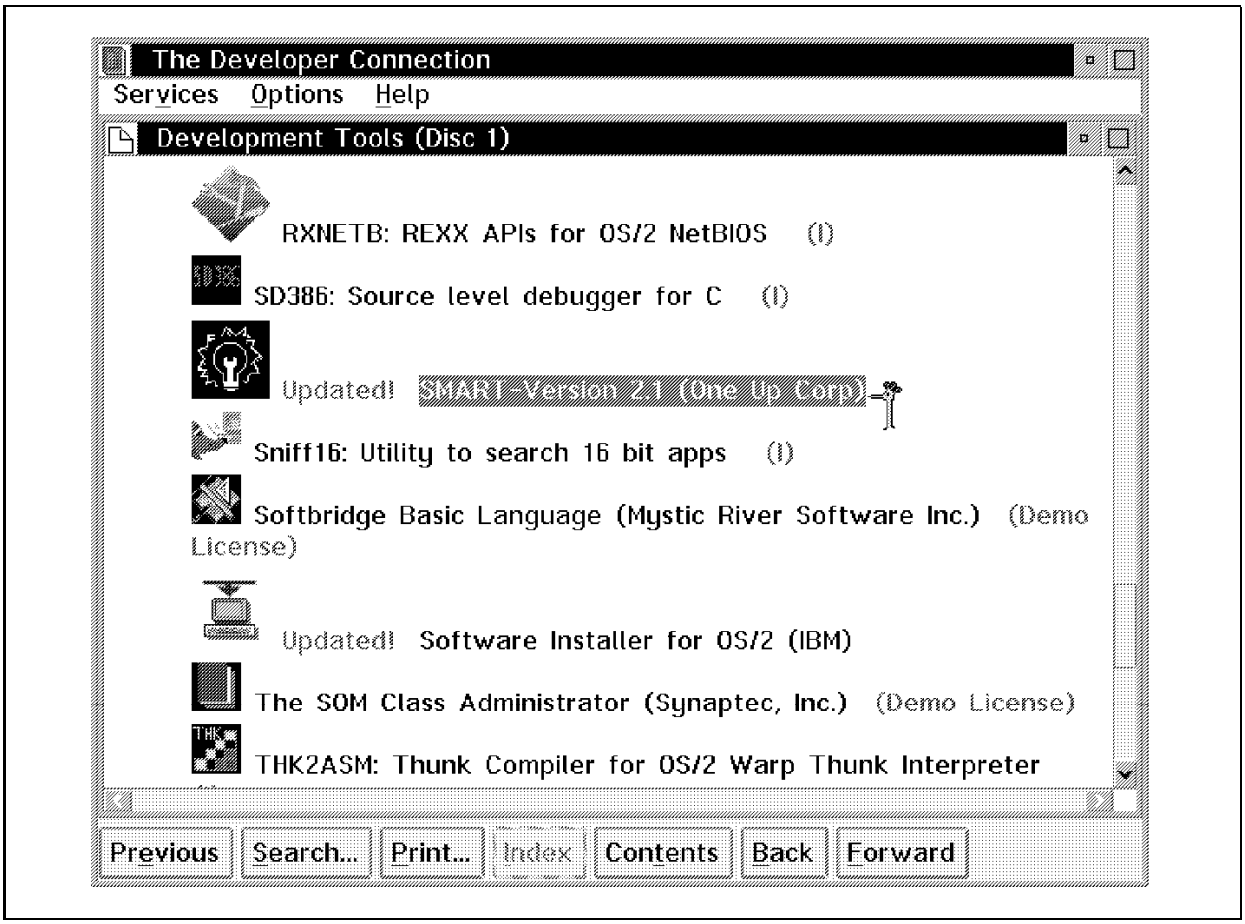


Figure 46. Development Tools in The Developer Connection for OS/2 Volume 10

5. After reading the information on SMART, you can double click on **Install me!**, shown in Figure 47 on page 50, to begin the installation of SMART.

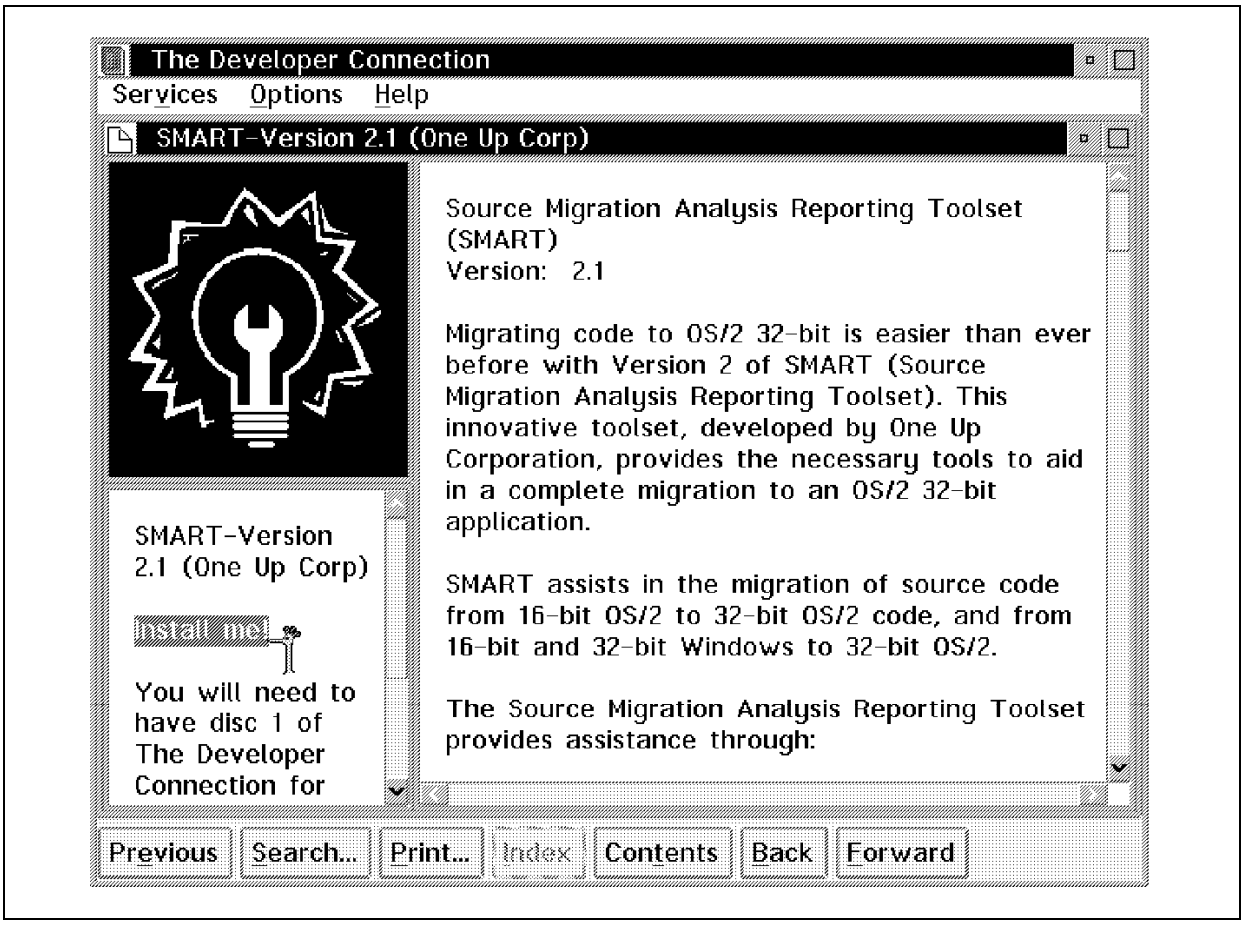


Figure 47. Information about SMART-Version 2.1 in The Developer Connection for OS/2 Volume 10

6. The SMART Installation window, shown in Figure 48 on page 51, will appear. Select the **Install** menu item.

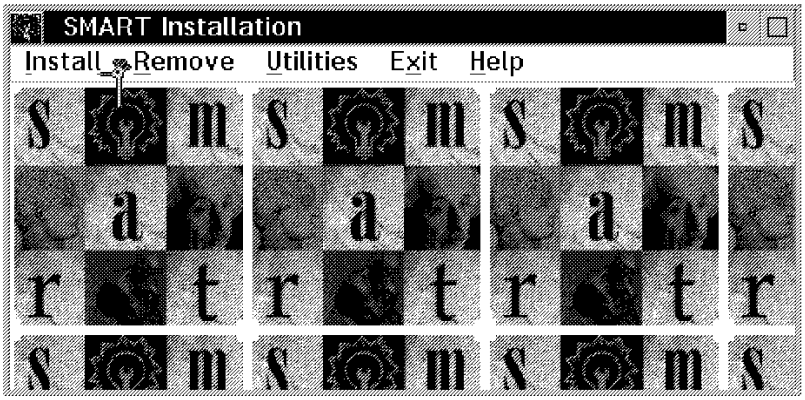


Figure 48. SMART Installation Window

7. The SMART Paths window will open to allow you to select the installation target drive and directory. The Installation Source should be your CD drive. Type the directory where you want SMART installed in the Target Installation box, shown in Figure 49

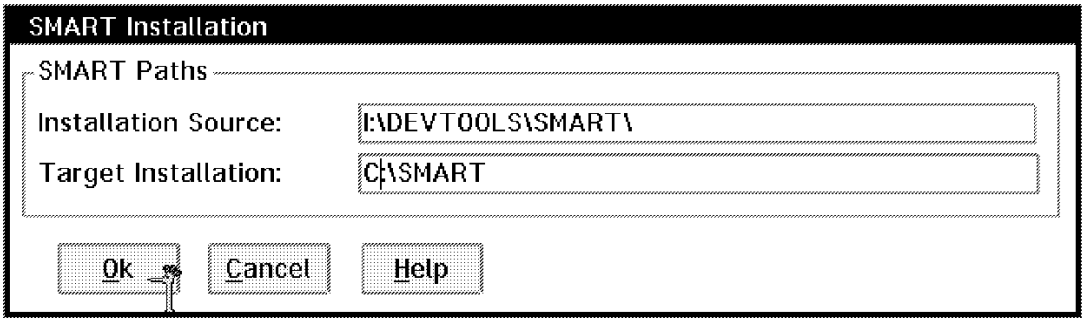


Figure 49. SMART Installation: Source and Target Paths

8. The Installation Setup Options window will open. You should install the Win32 -> OS/2 2.1 migration table, the SMART Editor, and the SMART Aux. Files, as shown in Figure 50 on page 52. If you will be using SMART to migrate applications for Windows 3.1 or OS/2 1.3, you can optionally, install those migration libraries.

   For performance reasons, it is recommended that you install SMART to your hard disk and not run it from the CD drive.

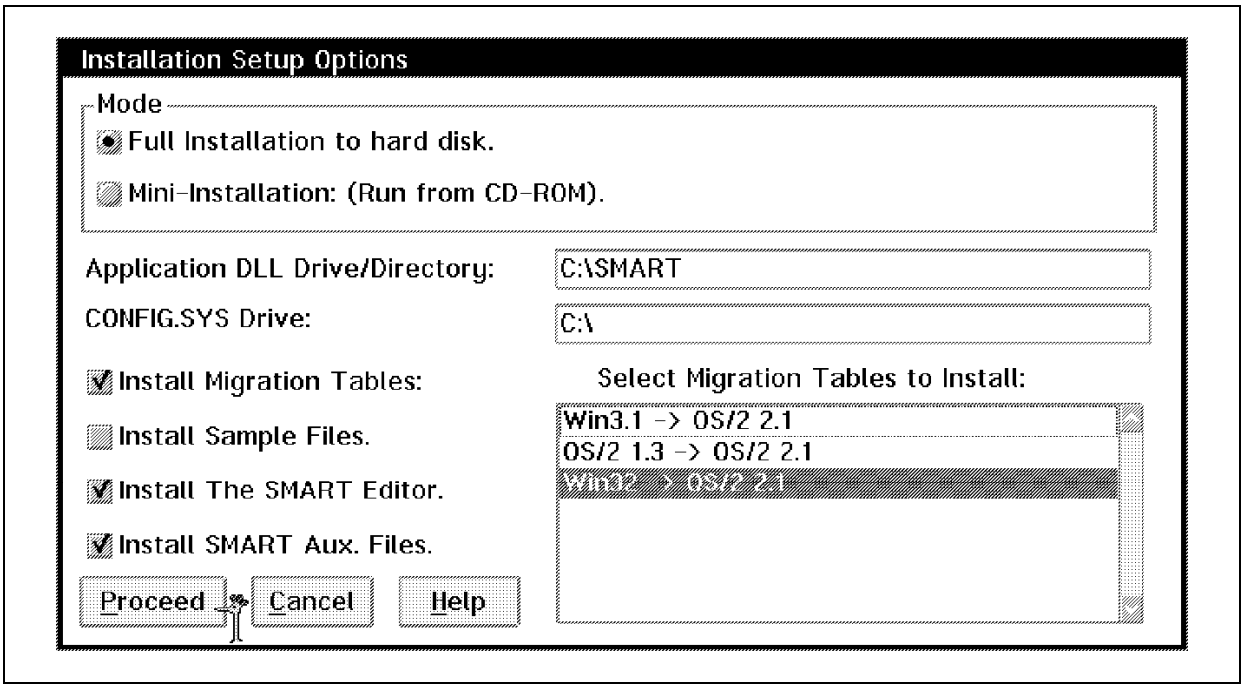


Figure 50. SMART Installation Setup Options

Note

The Application DLL drive/directory and CONFIG.SYS drive may vary from system to system. The default settings are appropriate.

9. After you have selected your installation options, select the **Proceed** pushbutton to begin copying files.
10. The Installing SMART window, shown in Figure 51, will appear while the files are copied to your hard drive.

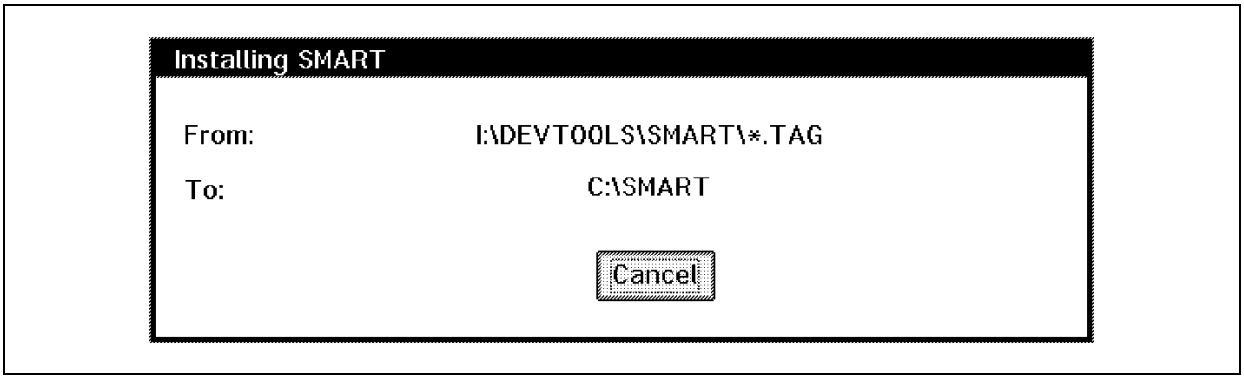


Figure 51. Installing SMART Window

11. After the files are copied, the CONFIG.SYS Maintenance window will appear, as shown in Figure 52 on page 53. Select **Ok=Continue** to allow SMART to modify your CONFIG.SYS file.

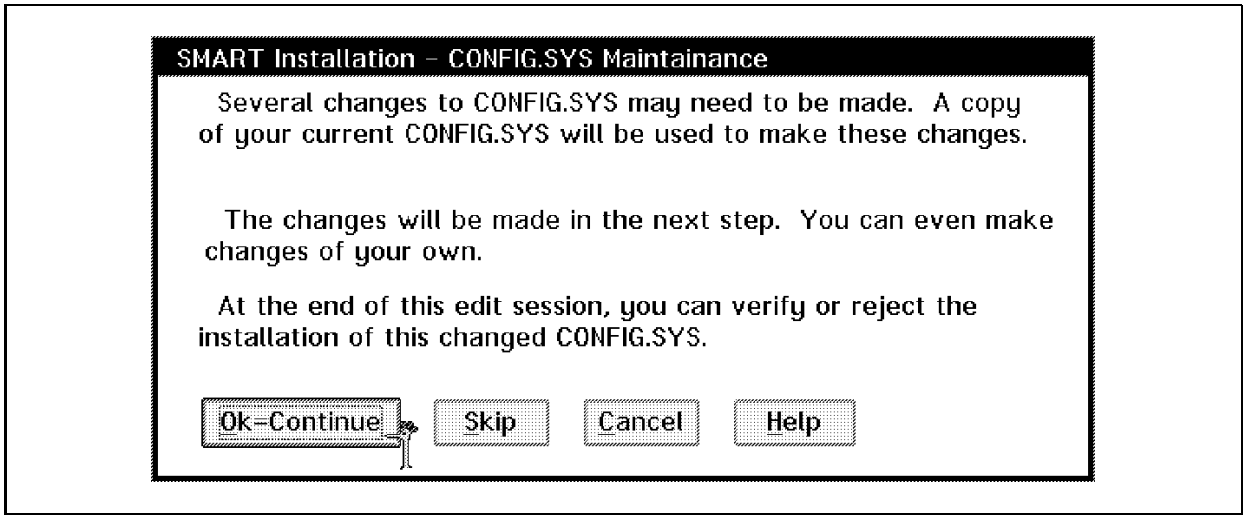


Figure 52. SMART Installation--CONFIG.SYS Maintenance

12. In most cases, no changes will be needed to CONFIG.SYS, and you will see the window shown in Figure 53. Select **Ok** to continue.

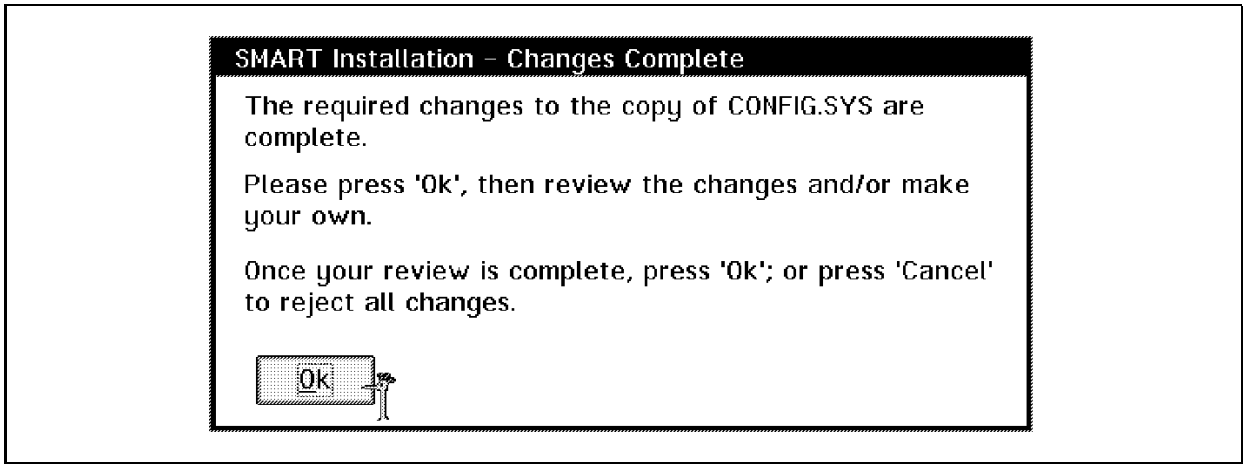


Figure 53. SMART Installation--Changes Complete

13. You will be left at the SMART Installation - edit CONFIG.SYS window, shown in Figure 54 on page 54. Select **Ok=Complete** to finish the installation.

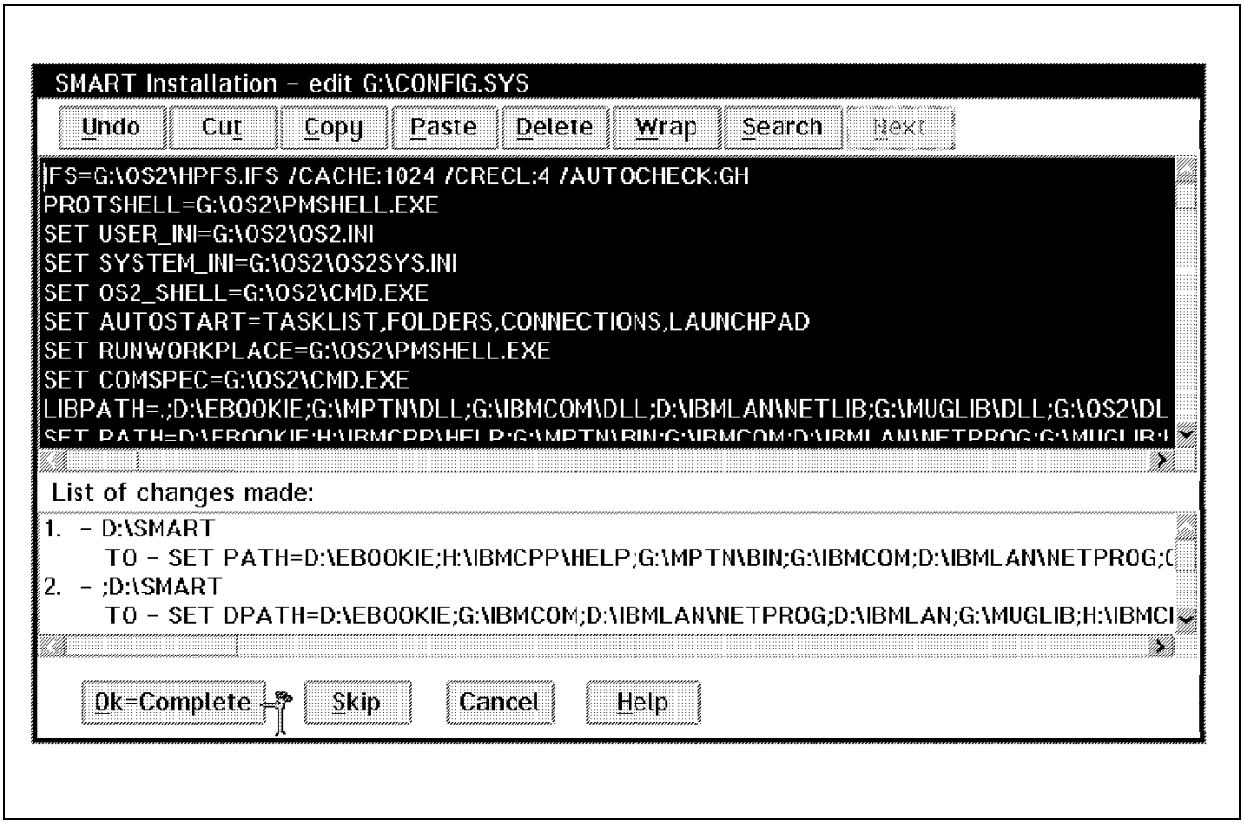


Figure 54. SMART Installation - Edit CONFIG.SYS

14. You will then see the SMART Installation - CONFIG.SYS window shown in Figure 55 on page 55. Select **Ok=Continue** to allow SMART to replace your CONFIG.SYS with the modified version.

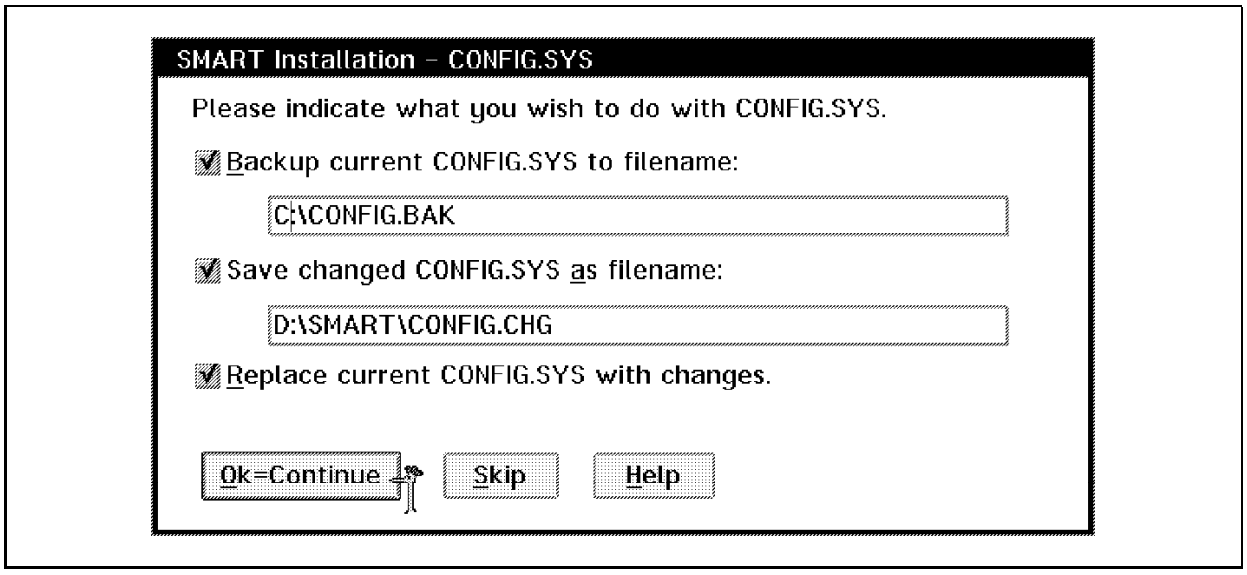


Figure 55. SMART Installation - CONFIG.SYS

15. The SMART Installation - Completed window, shown in Figure 56, will appear. After reading it, press **Ok** to close it.

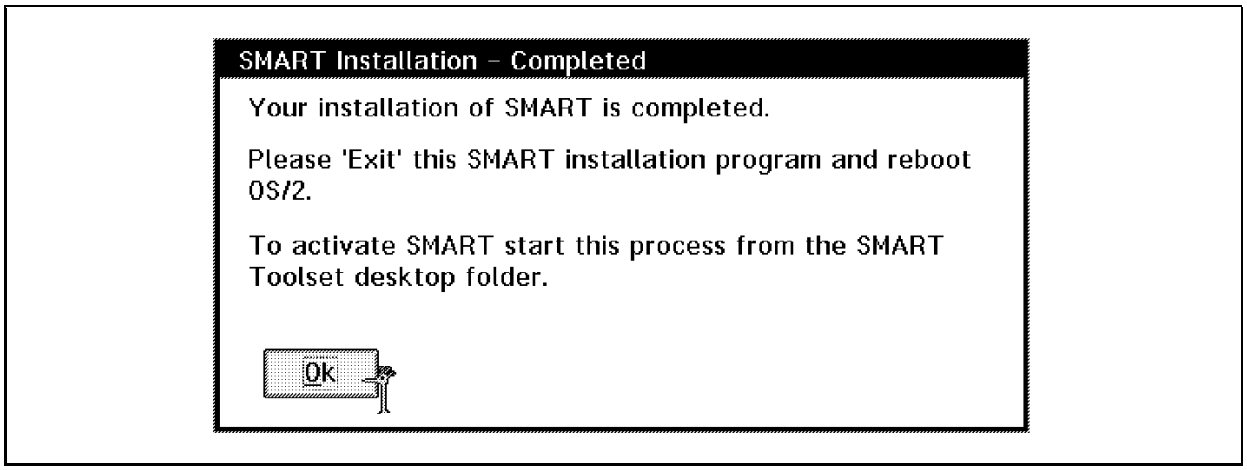


Figure 56. SMART Installation - Completed

16. You will return to the main SMART window shown in Figure 48 on page 51. Select **Exit** from the menu to close the installation program.
17. Shut down your computer and reboot to enable the changes SMART has made to your machine.

Note

You should reboot your computer even if no changes were made to CONFIG.SYS.

2.4.2 Where to Learn More About SMART

You can find more information about SMART in the SMART2 Toolset folder, which is created during SMART installation. The folder is shown in Figure 57.

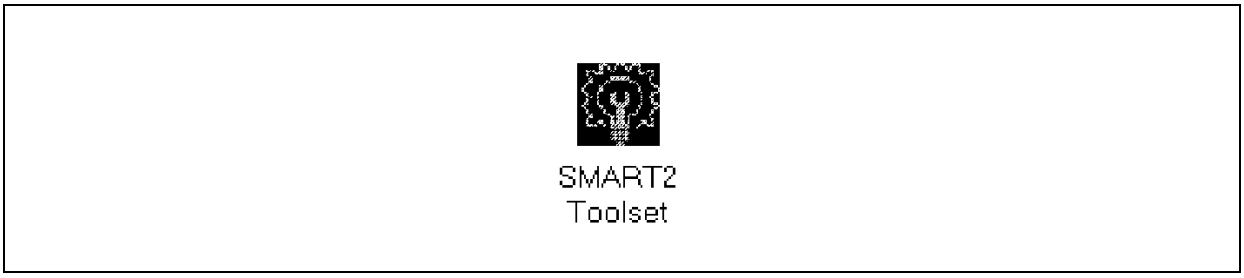


Figure 57. SMART2 Toolset Folder on the Desktop

Double click on the folder to open it. The folder's contents are shown in Figure 58.

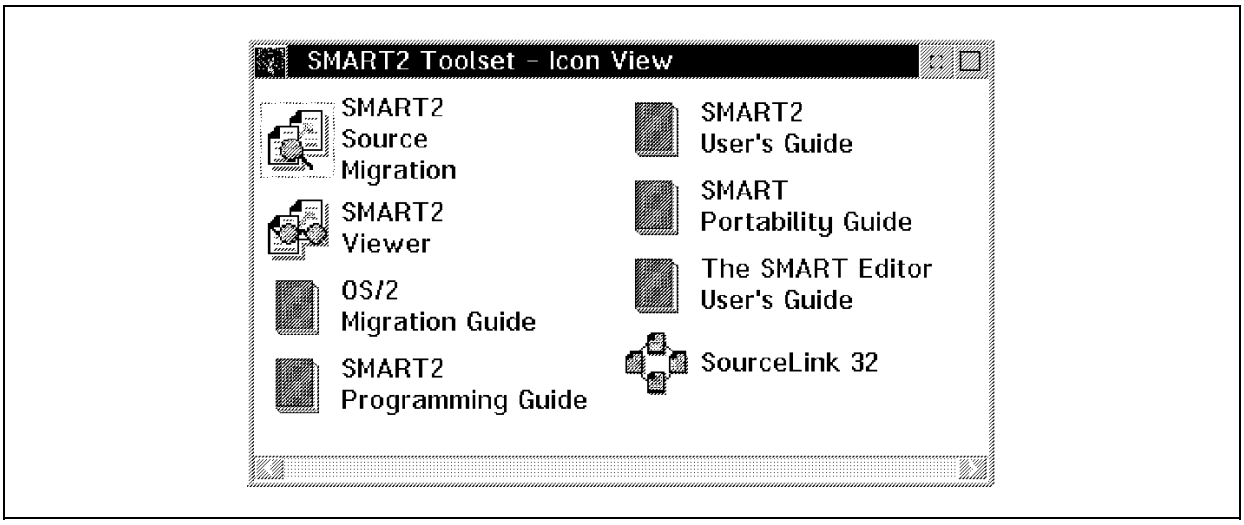


Figure 58. SMART2 Toolset Folder Contents

2.5 VisualAge C++

You will need to select and install a C/C++ compiler for OS/2 that can be used to build Open32 applications.

This section describes the installation of the VisualAge C++ Trial Version that is included on The Developer Connection for OS/2 Volume 10.

2.5.1 Installing VisualAge C++

VisualAge C++ is the newest release of IBM's C/C++ compiler for for OS/2. A sixty-day trial copy can be found on The Developer Connection for OS/2 Volume 10.

To install VisualAge C++:

1. Start the The Developer Connection for OS/2 Catalog by following the steps outlined in 2.1.2, “Starting The Developer Connection for OS/2 Volume 10” on page 22.
2. The catalog of The Developer Connection for OS/2 Products will then be displayed, as shown in Figure 59 on page 58.
3. Double click on **Compilers (Disc 4)**, shown highlighted in Figure 59 on page 58. You will be shown the list of compilers shown in Figure 60 on page 59.

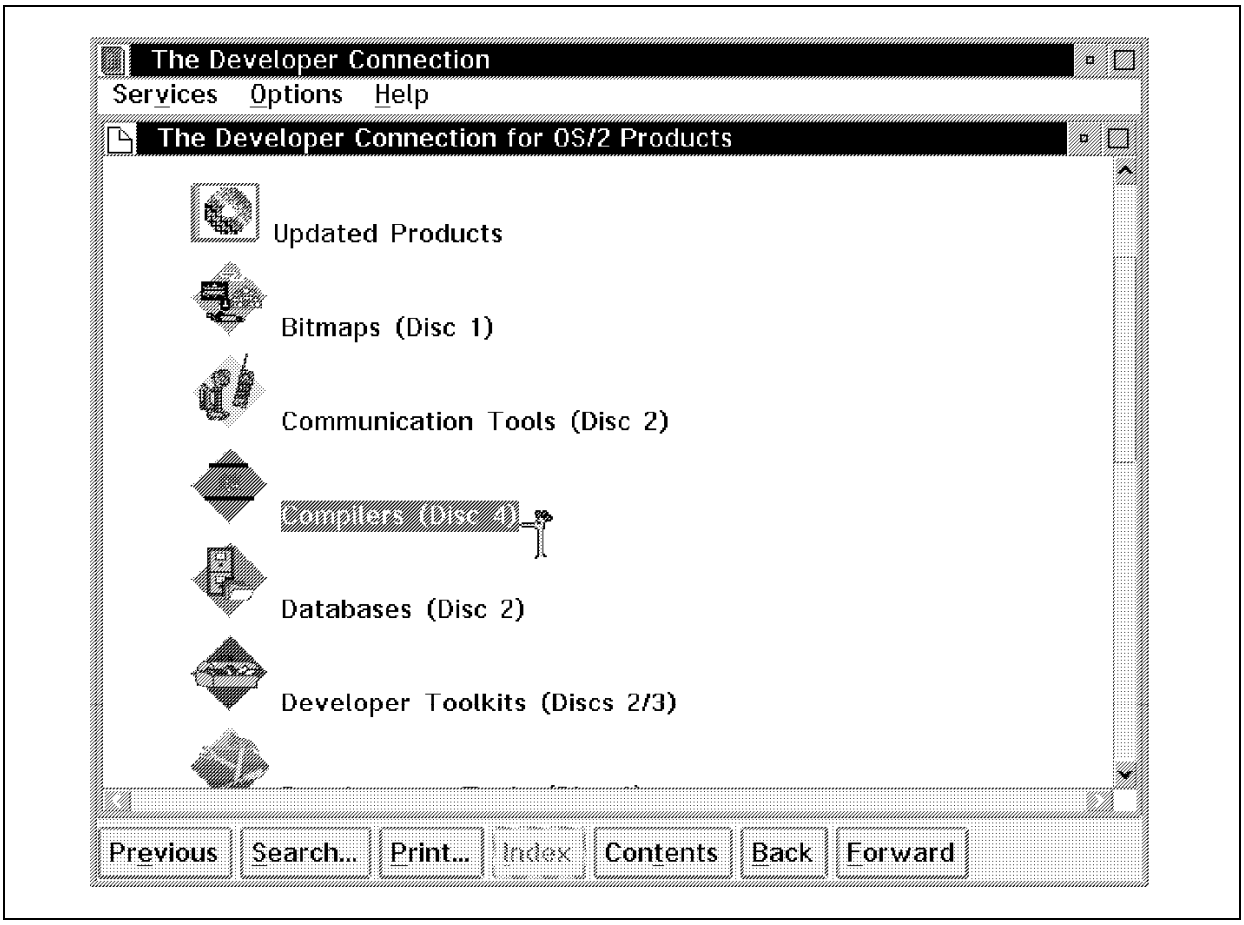


Figure 59. The Developer Connection for OS/2 Volume 10 Catalog

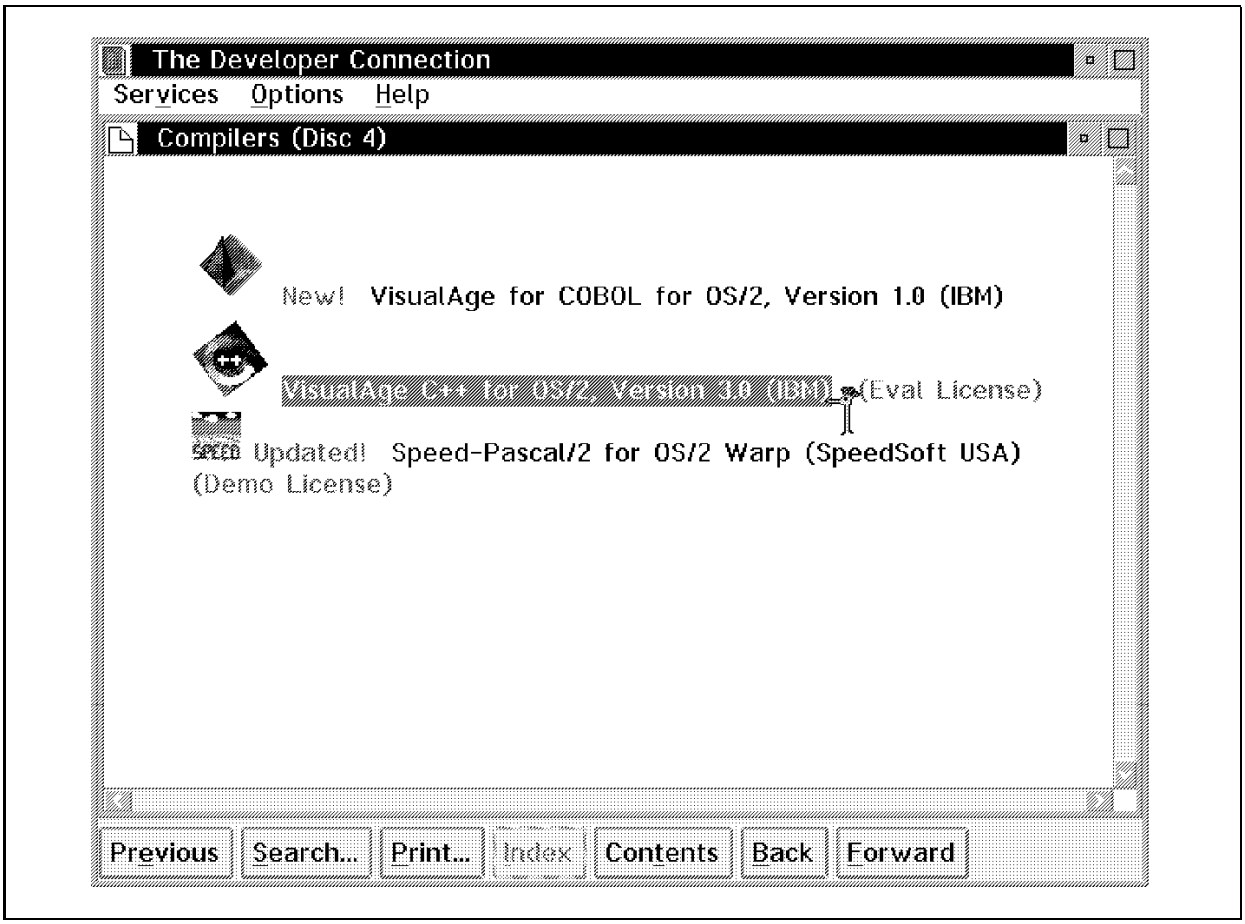


Figure 60. Compilers Available on The Developer Connection for OS/2 Volume 10

4. Double click on **Visual Age C++ for OS/2, Version 3.0 (IBM)**, shown in Figure 60, and the information about VisualAge C++ will be displayed as shown in Figure 61 on page 60.

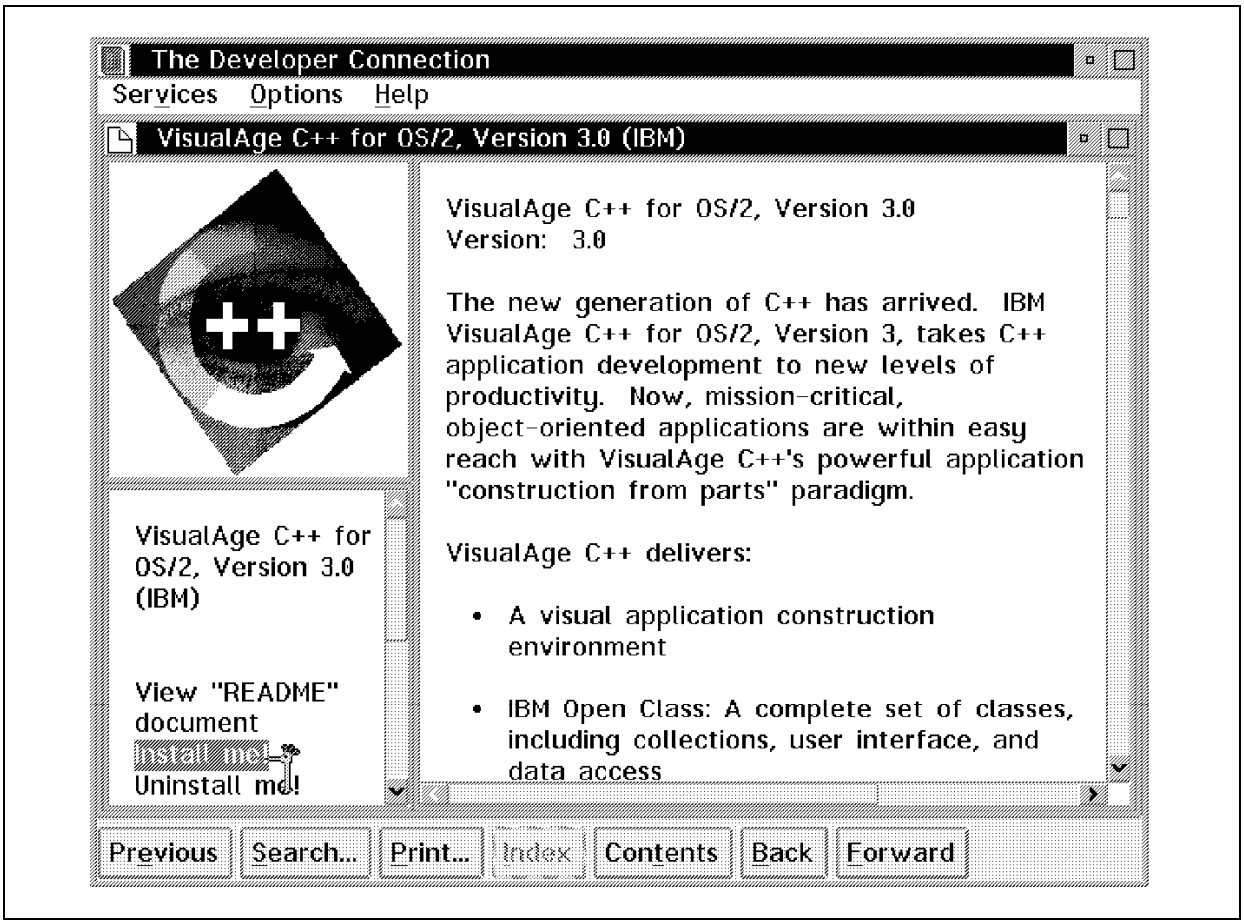


Figure 61. Information about VisualAge C++

5. After reading the information about VisualAge C++ for OS/2, you can double click on **Install me!** in the lower left corner of the information panels to start the VisualAge C++ installation program.

   You will be presented with the Welcome to VisualAge C++ Version 3.0 as shown in Figure 63 on page 61.

   If the Disc request dialog, shown in Figure 62 on page 61, appears, it could be because the disc in the CD-ROM drive is not DISC 4 of The Developer Connection for OS/2 Volume 10 or the drive letter is not a valid CD-ROM drive. Make sure that Disc 4 and the drive letter are correct, then select the **OK** pushbutton of the Disc request dialog to start the installation.

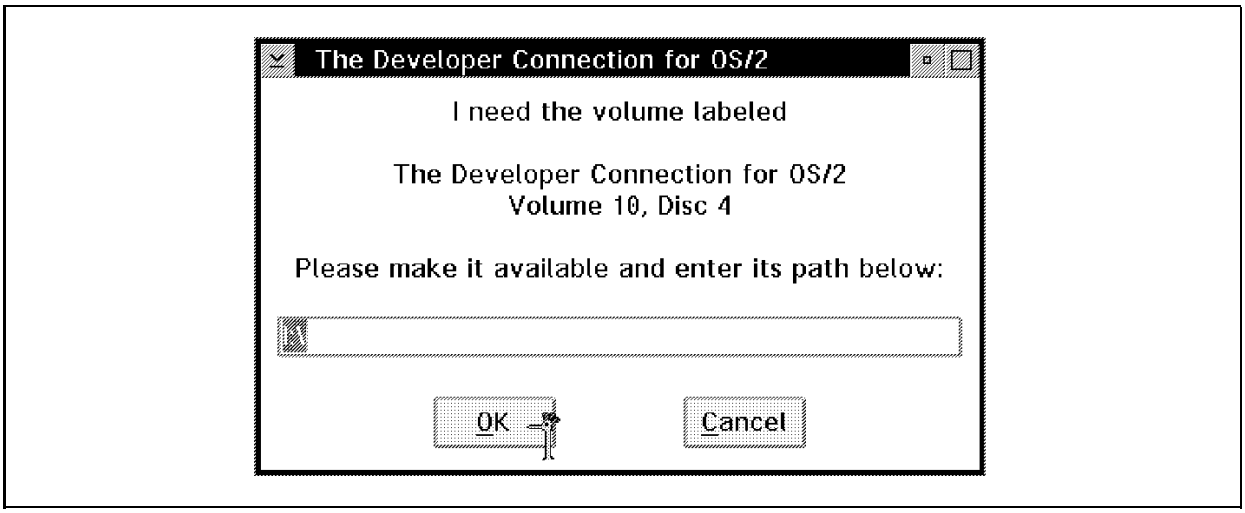


Figure 62. Disc Request Screen

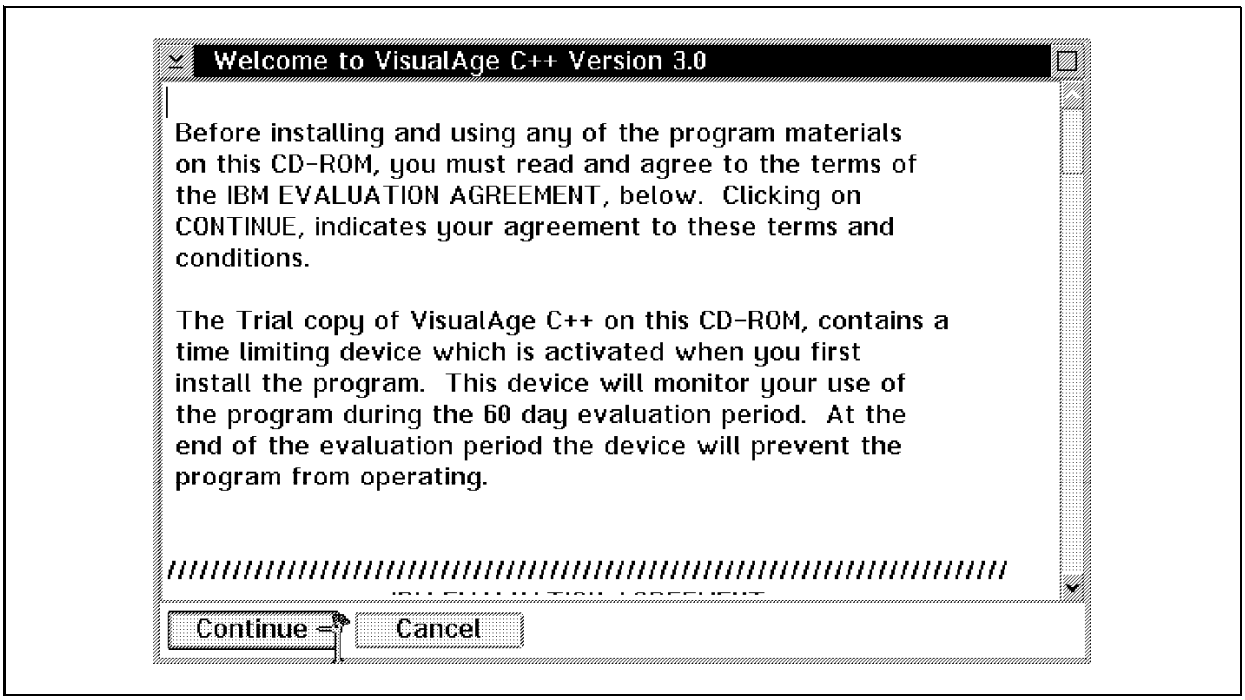


Figure 63. Welcome to VisualAge C++

6. You should read the IBM Evaluation Agreement in the Welcome to VisualAge C++ Version 3.0 window before proceeding. If you agree to the terms of the agreement, select **Continue**.

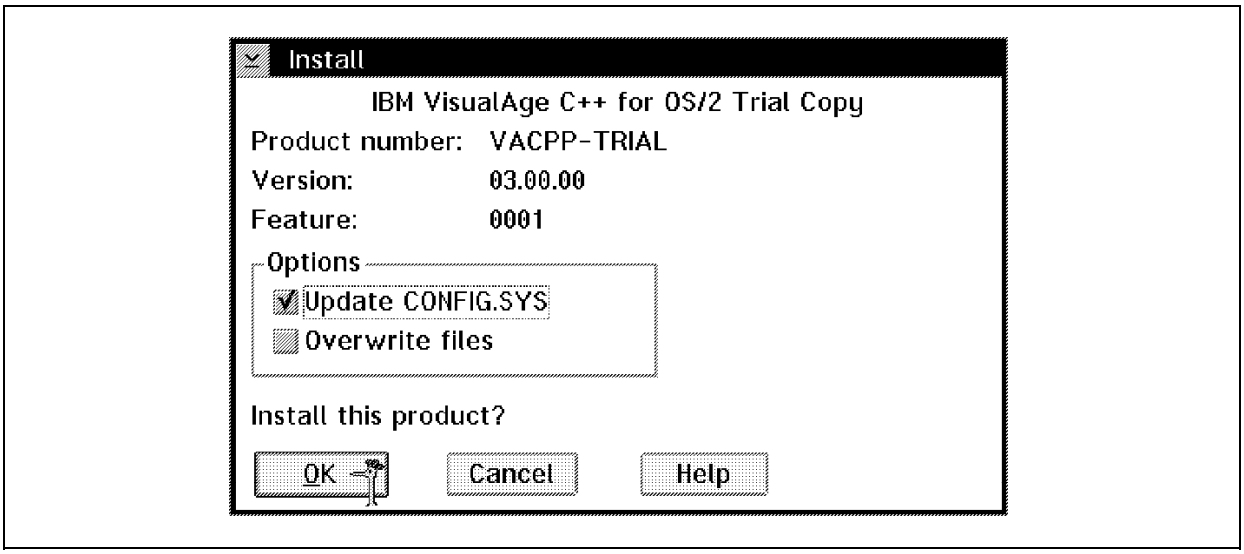


Figure 64. VisualAge C++ Install

7. The Install dialog will be displayed. It gives you the options of updating CONFIG.SYS and overwriting files. The default options of Update CONFIG.SYS and do not Overwrite files are recommended. When the options are set the way you want, select **OK** on the Install dialog and the **Installation-directories** dialog will appear, as shown in Figure 65 on page 63.

   The **Install-directories** panel, shown in Figure 65 on page 63, allows you to select the components of VisualAge C++ to be installed. You can also specify where you want the files to be placed on your hard disk.

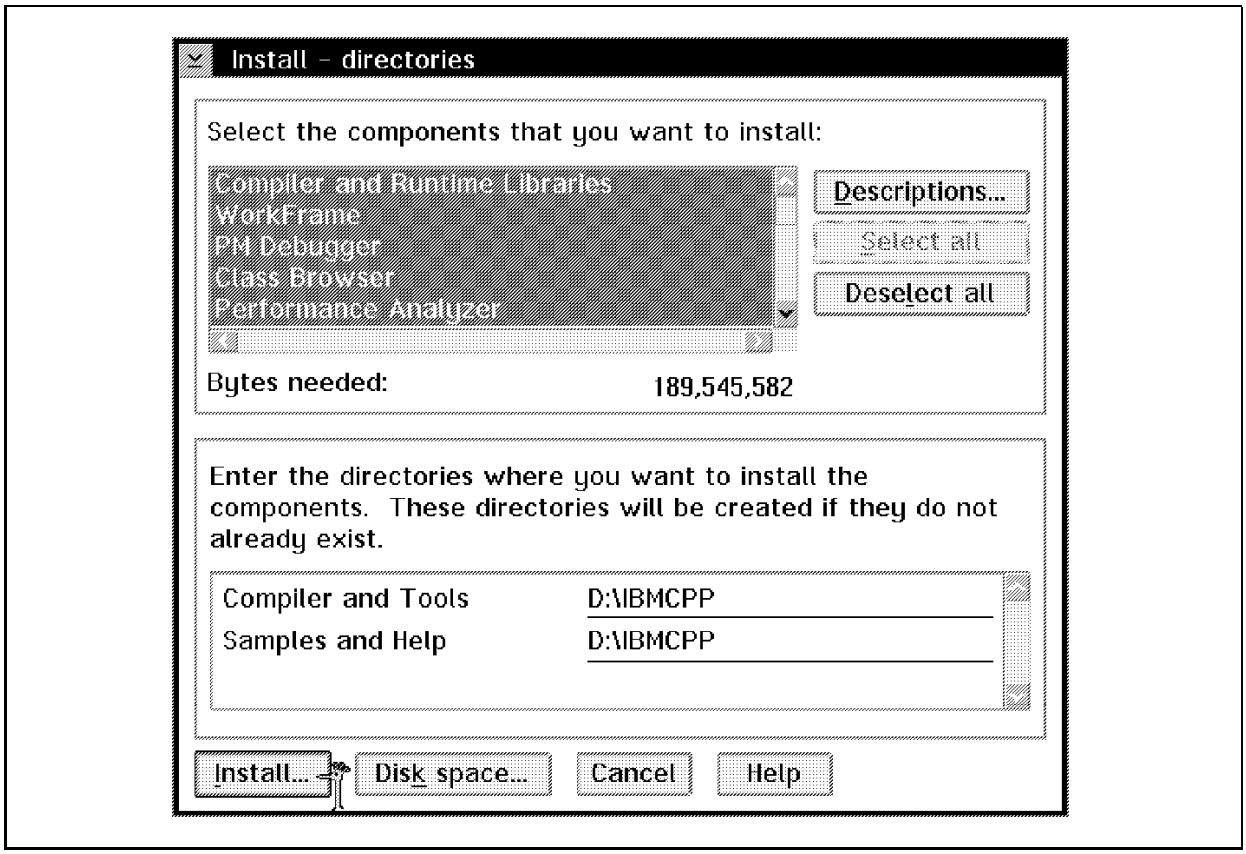


Figure 65. VisualAge C++ Install-Directories

The following list outlines the contents of VisualAge C++ and gives recommendations on what portions to install for Open32 development.

Item Name	Recommendation
Compiler and Runtime Libraries	Required
WorkFrame	Recommended
PM Debugger	Recommended
Class Browser	Not required
Performance Analyzer	Not required
Visual Builder	Not required
Data Access Builder	Not required
Editor	Recommended only if you have not selected an editor for the OS/2 platform

IBM Open Class Library	Not required
Standard Class Library	Not required
Documentation	Recommended
Samples	Not required
Warp Toolkit Development Tools	Not Required
Warp Toolkit Information	Not Required
Warp Toolkit Headers & Libraries	Not Required
Warp Toolkit Sample Programs	Not required
Warp Toolkit Multimedia Bitmaps	Not required

You can reduce the items to be installed by deselecting them in the Select the components that you want to install list box. This is done by a single click of mouse button one on the items to be deselected.

The minimum installation (including only required and recommended items) requires about 80MB as shown in Figure 66.

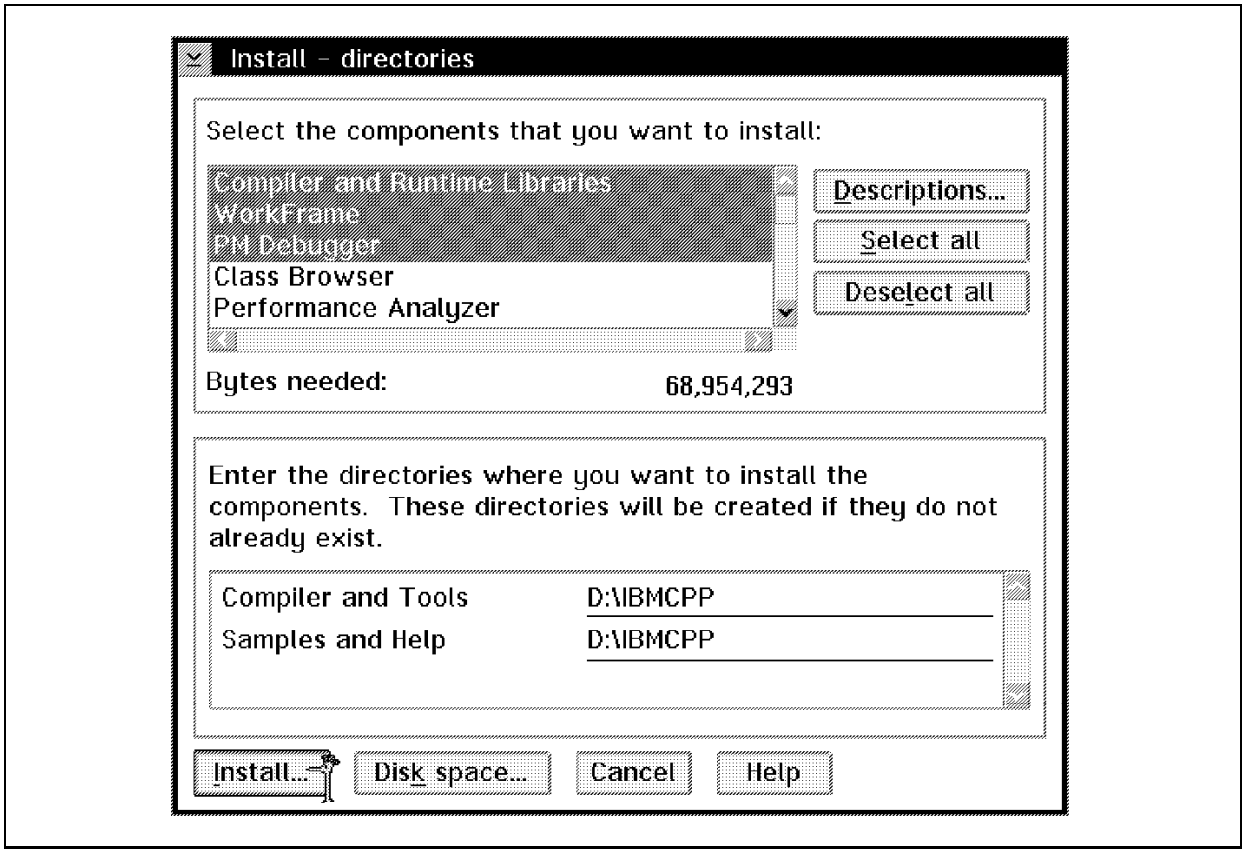


Figure 66. VisualAge C++ Minimum Install

Complete installation of VisualAge C++ requires about 200MB of disk space.

8. After you have selected the components you want to install you may want to change the install to directories. Edit the directory names on the Install - directories dialog or click on **Disk space...** to view the free space on your hard drives and optionally choose one for installation. The Disk space dialog is shown in Figure 67.

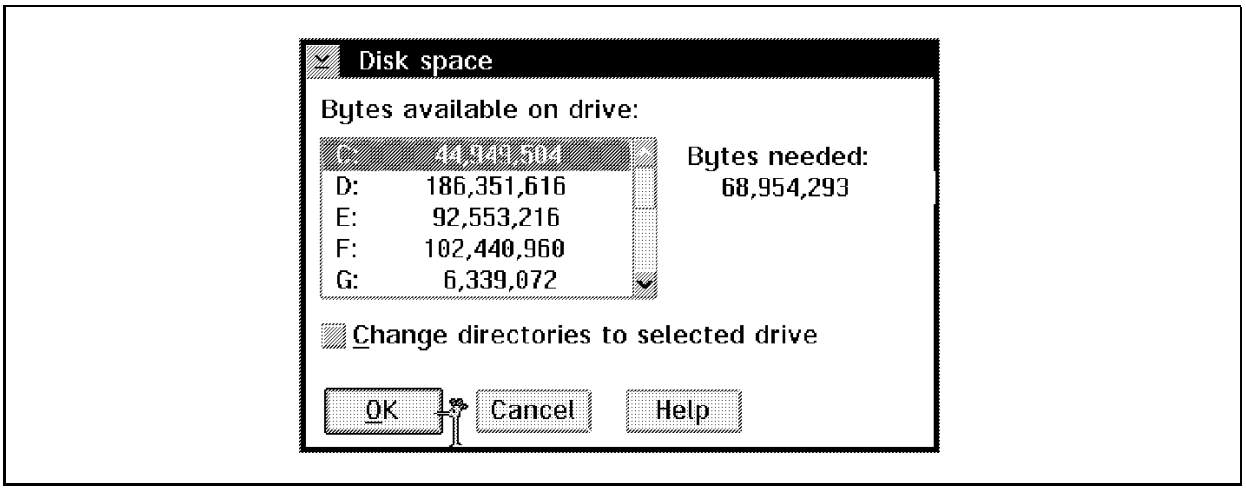


Figure 67. VisualAge C++ Disk Space

The Disk space dialog, shown in Figure 67, shows you the free space available on your hard drives. You can change the drive VisualAge C++ will be installed to by highlighting the drive in the list and check the **Change directories to selected drive** box. Click on **OK** to select the drive and return to the Install - directories dialog.

9. After you have selected which parts of VisualAge C++ to install and where to put them on your computer, select the **Install...** pushbutton to begin copying files to your computer.

   The Install - progress window, shown in Figure 68 on page 66 will appear to display the files being copied and to show the progress of the installation.

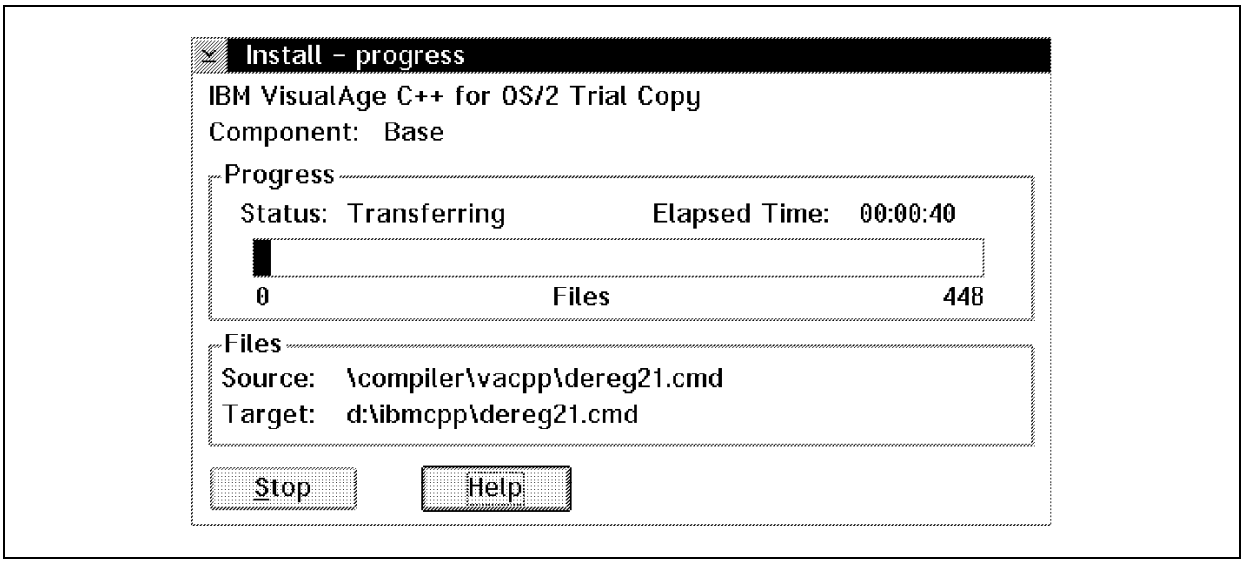


Figure 68. VisualAge C++ Install - Progress

After the copying process is finished, the dialog shown in Figure 69 will appear to notify you that VisualAge C++ was successfully installed on your computer.

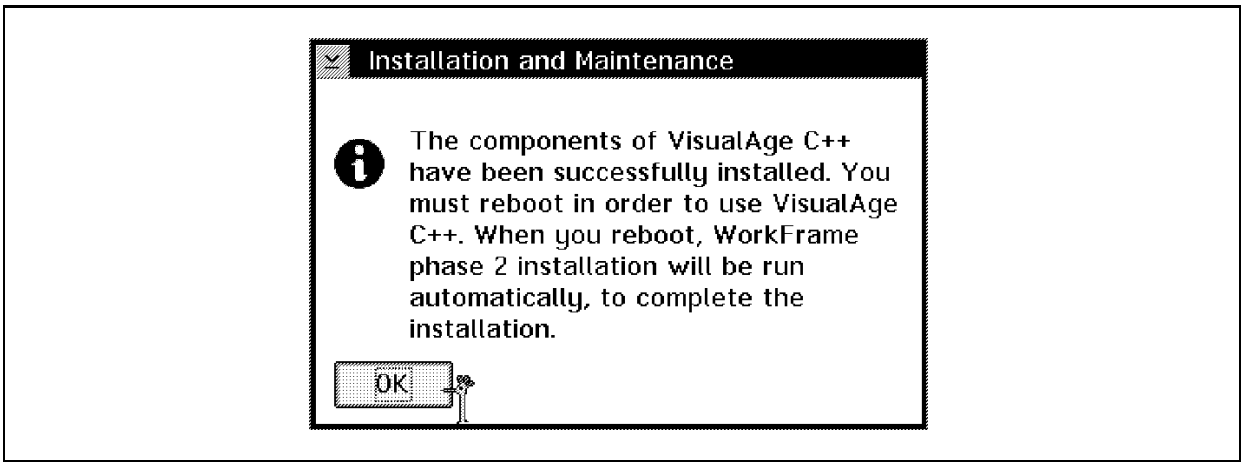


Figure 69. VisualAge C++ Successfully Installed

10. Select the **OK** pushbutton on the Installation and Maintenance window as shown in Figure 69 to close the message window.
11. Select the **Exit** pushbutton on the IBM VisualAge C++ Trial Copy window as shown in Figure 70 on page 67 to close the installation window.

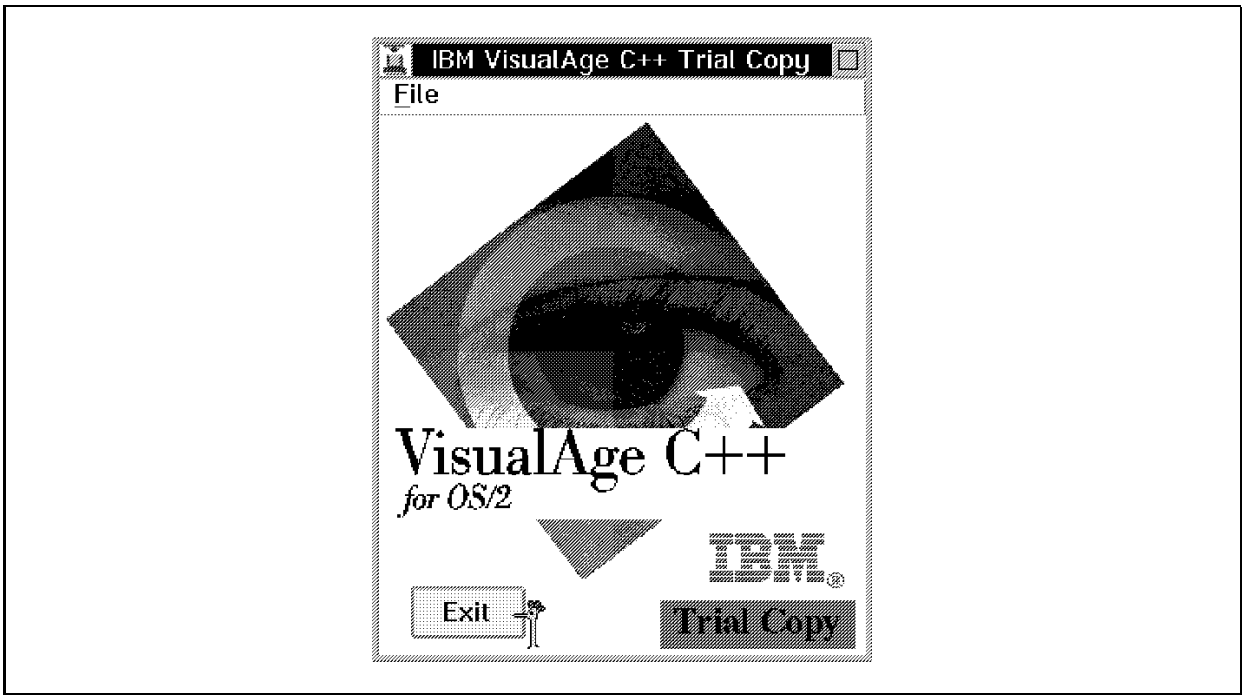


Figure 70. VisualAge C++ Installation Window

12. Shut down OS/2 and restart your computer.

    The second phase of installation will be started automatically in a windowed command as shown in Figure 71 on page 68.

```
Wincreateobject C main template succeeded
Creating Project Smarts object. . .
Wincreateobject Project Smarts succeeded
Updating 18 shell projects (this will take a few minutes)...
1 done.
2 done.
3 done.
4 done.
5 done.
6 done.
7 done.
8 done.
9 done.
10 done.
11 done.
12 done.
13 done.
14 done.
15 done.
16 done.
17 done.
18 done.
All shell projects updated.
Populating Smarts catalog with entries...
```

Figure 71. Install Phase 2

The windowed command will close automatically.

Figure 72. VisualAge C++ Folder on Desktop

13. After OS/2 finishes booting, open the **VisualAge C++ Trial Copy** by double clicking on its icon on the desktop, shown in Figure 72.

    The VisualAge C++ folder, shown in Figure 73 on page 69, will appear.

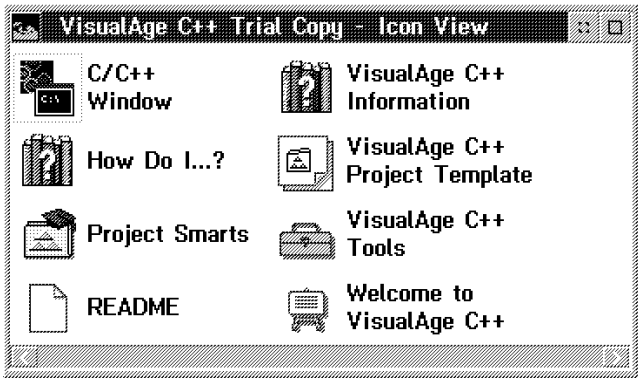


Figure 73. VisualAge C++ Folder

2.5.2 Where to Learn More About VisualAge C++

You can find on-line documents in the VisualAge C++ folder.

1. Open the **VisualAge C++ Information** folder by double clicking on its icon, as shown in Figure 73.
2. The VisualAge C++ Information folder contains many documents, as shown in Figure 74 on page 70, which you may want to study at your convenience.

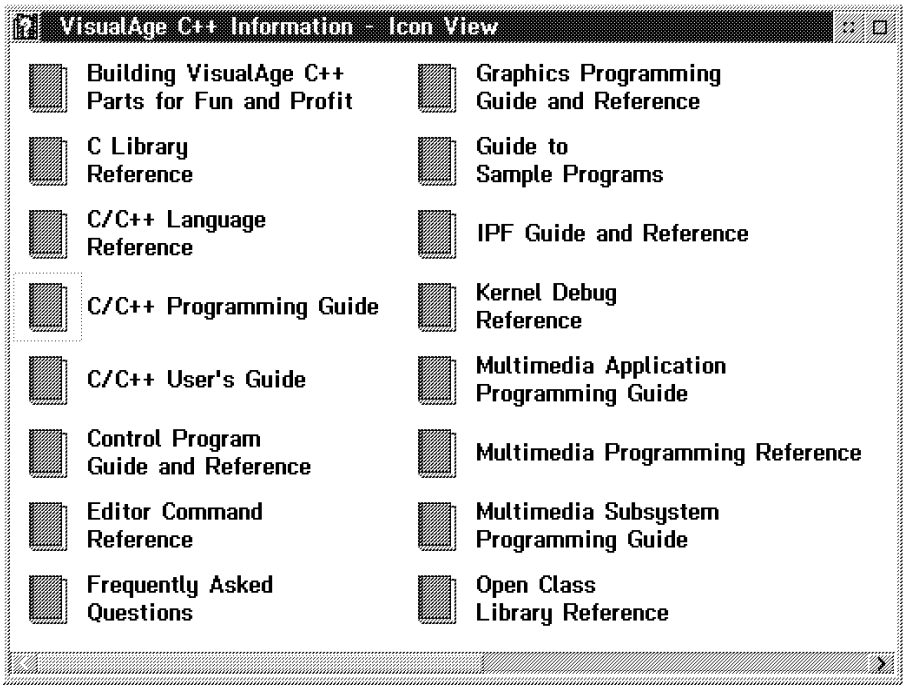


Figure 74. VisualAge C++ Information Folder

2.6 Developer API Extensions

The Developer API Extensions are a set of dynamic link libraries that provide Open32 applications with procedures that are functionally equivalent to the standard Win32 APIs. You must install these extra DLLs on your OS/2 Warp or OS/2 Warp Connect computer to run the Open32 applications you develop. OS/2 Warp Version 4 has the extensions built-in, so if you are running OS/2 Warp Version 4, you can skip this step and proceed to the first sample Open32 program found in Chapter 3, “Howdy, World!” on page 79.

2.6.1 Installing Developer API Extensions

The Developer API Extensions are shipped on The Developer Connection for OS/2 Volume 10 for installation on OS/2 Warp and OS/2 Warp Connect. If you are using OS/2 Warp Version 4, you do not need to install the Developer API Extensions; it is built into the base operating system. Installation requires approximately 2.5MB on your OS/2 hard drive.

To install Developer API Extensions:

1. Start The Developer Connection for OS/2 Catalog by following the steps outlined in 2.1.2, “Starting The Developer Connection for OS/2 Volume 10” on page 22. The catalog of The Developer Connection for OS/2 Products will be displayed as shown in Figure 75 on page 71.
2. Double click on **IBM OS/2 (Discs 1/2/3)** from The Developer Connection for OS/2 Volume 10 main menu shown in Figure 75. This will open the list of IBM OS/2 products shown in Figure 76 on page 72.

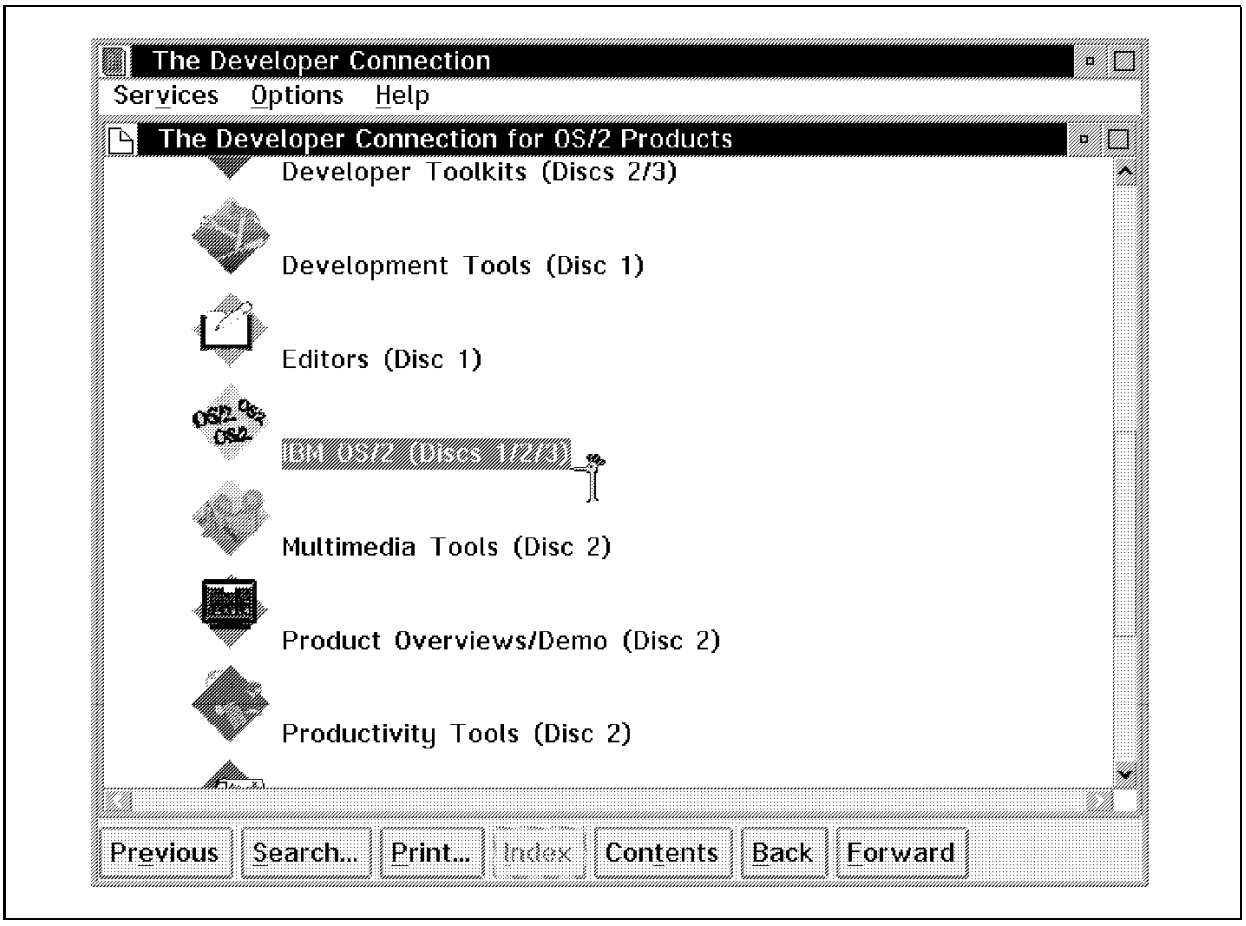


Figure 75. The Developer Connection for OS/2 Volume 10 Catalog

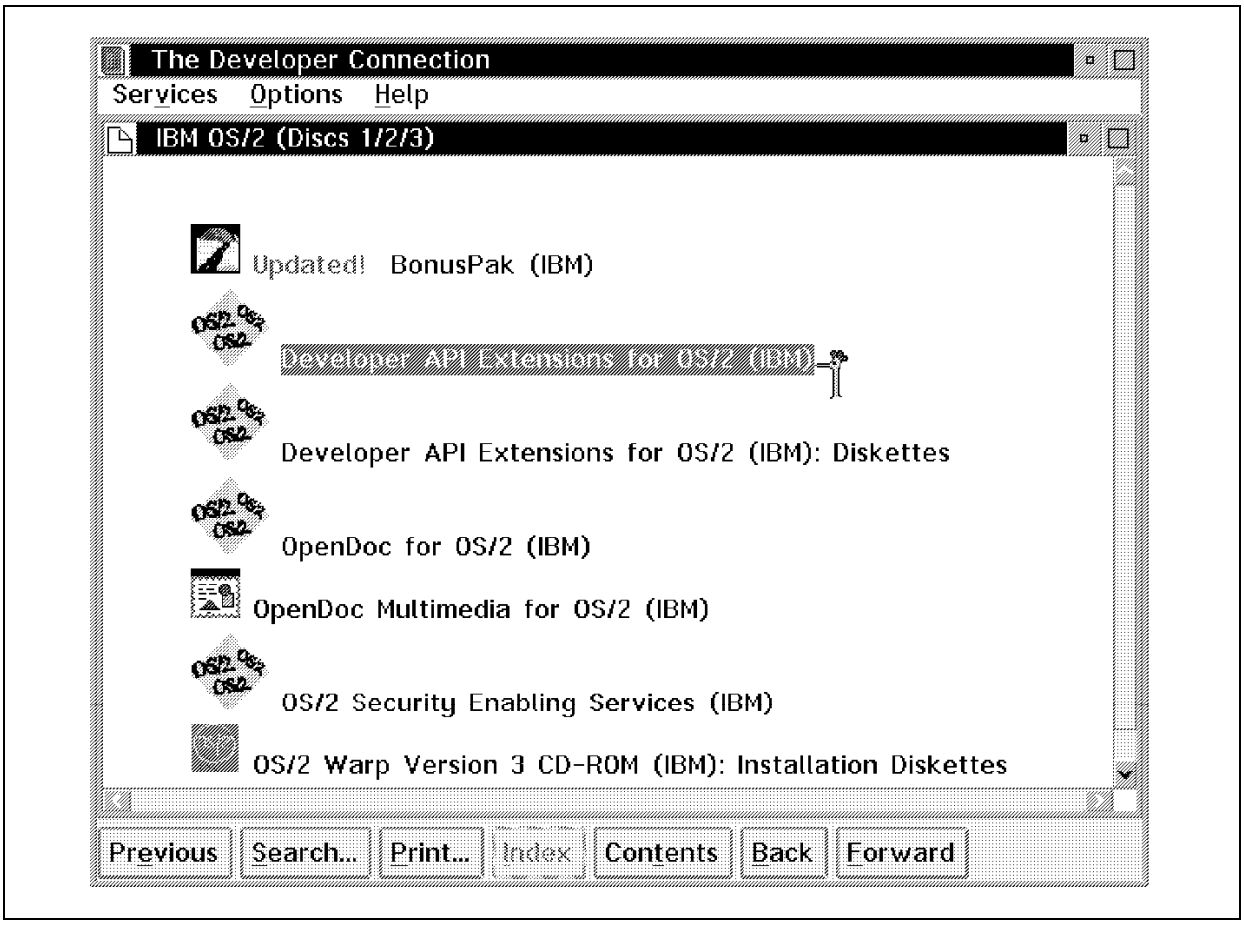


Figure 76. IBM OS/2 Products Available on The Developer Connection for OS/2 Volume 10

3. Double click on **Developer API Extensions for OS/2 (IBM)** from the menu shown in Figure 76. You will see the information panels on Developer API Extensions for OS/2 (IBM), shown in Figure 77 on page 73.

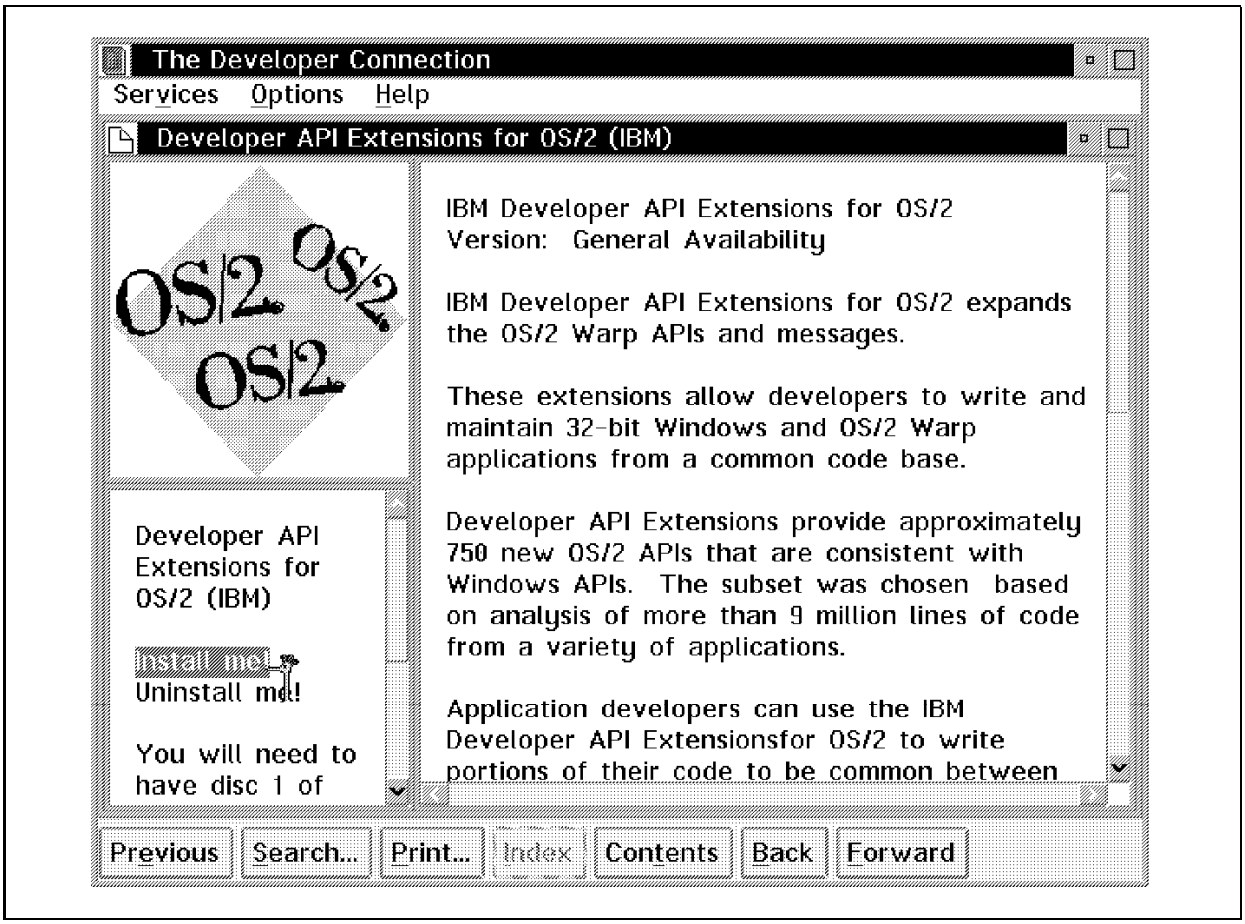


Figure 77. Developer API Extensions Information Screen

4. After reading the information about the IBM Developer API Extensions for OS/2, you can double click on **Install me!** to start the Developer API Extensions installation program.
5. Developer API Extensions installation will display the install dialog window shown in Figure 78 on page 74. Press **OK** to continue with the installation.

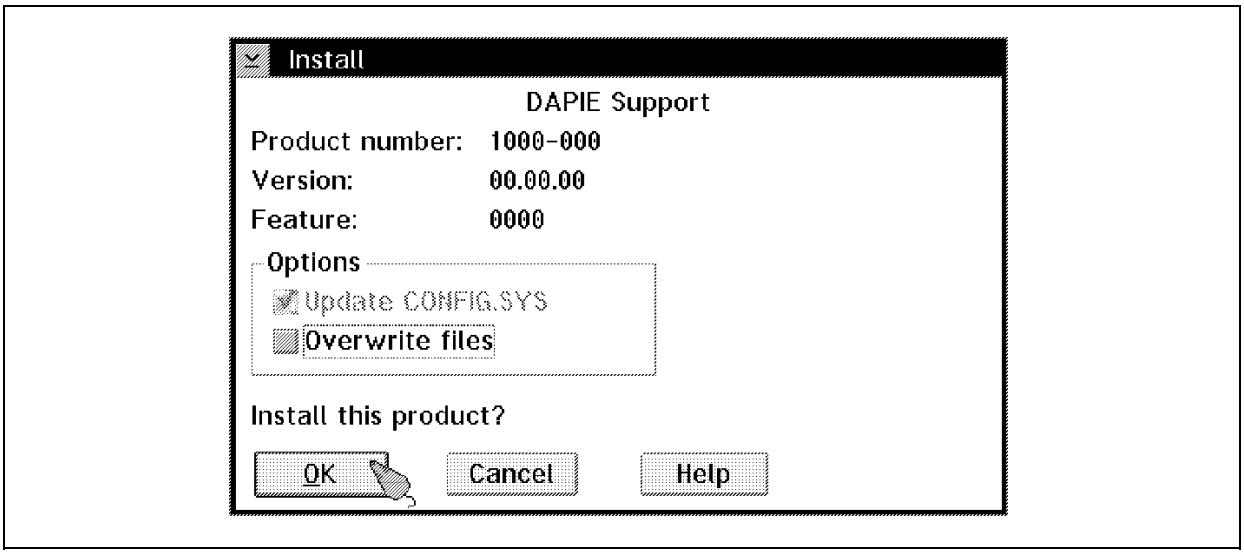


Figure 78. Developer API Extensions Install

The Install - directories window will appear as shown in Figure 79 on page 75.

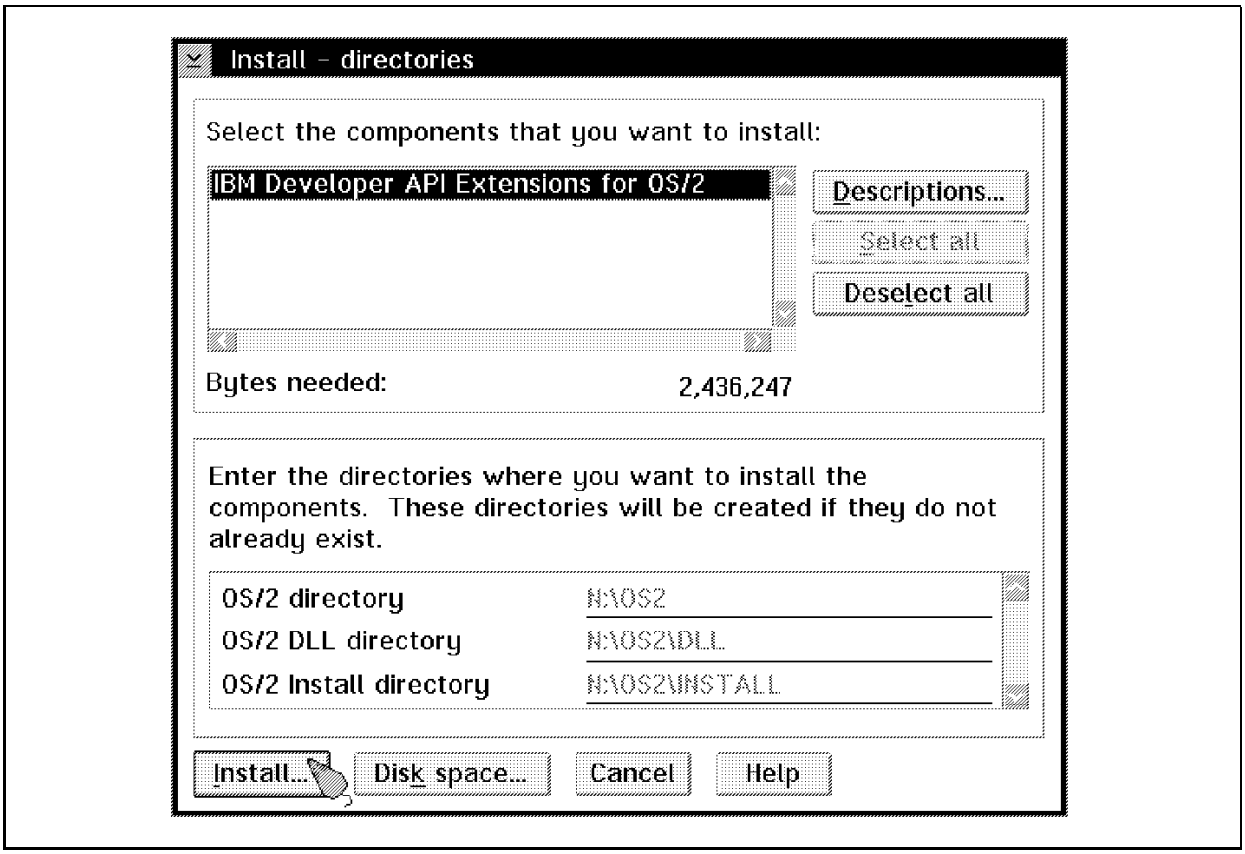


Figure 79. Developer API Extensions Install - Directories

6. Select **IBM Developer API Extensions for OS/2** in the Select the components that you want to install: list box.

   Note

   The Developer API Extensions will be installed to your OS/2 system directories by default. You should not change the location where the files will be copied.

7. Select the **Install...** pushbutton to begin copying files.

   You will see the Install - progress window as shown in Figure 80 on page 76 during the installation.

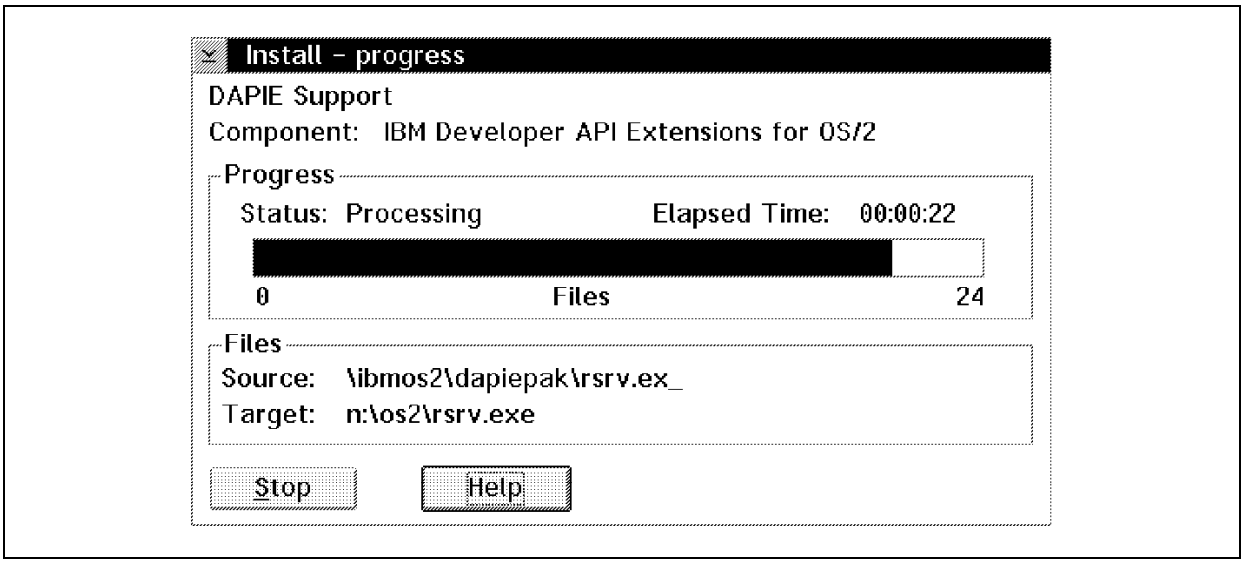


Figure 80. Developer API Extensions Install - Progress

8. Once all the files have been copied you will be informed of the successful installation with the Installation Status window as shown in Figure 81. Select **OK** to close it.

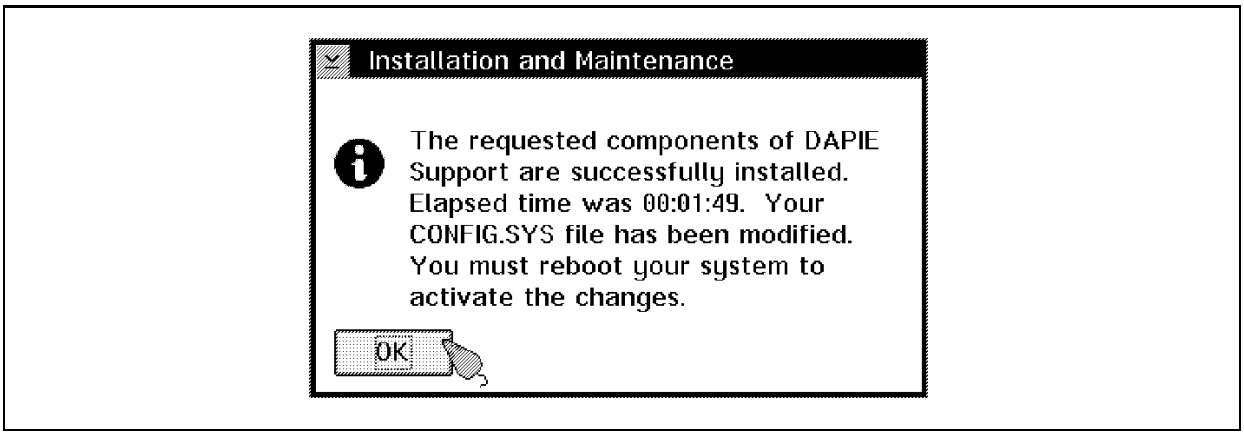


Figure 81. Developer API Extensions Successfully Installed

9. Close the installation program by selecting the **Exit** pushbutton, as shown in Figure 82 on page 77.

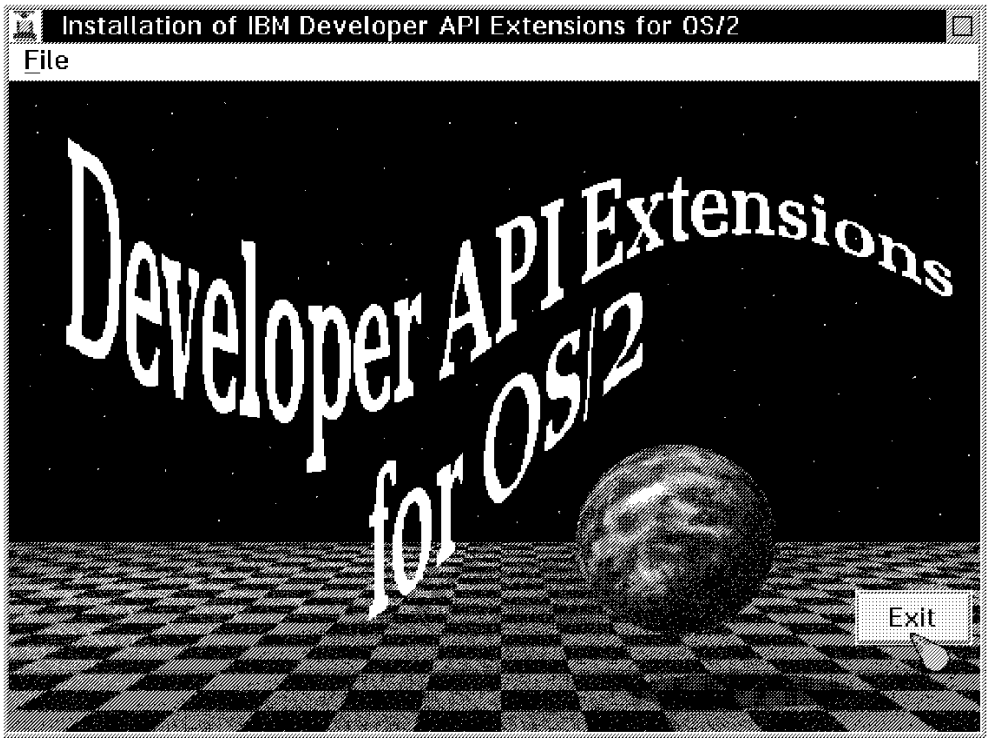


Figure 82. Developer API Extensions Installation

10. After the installation has completed, you will need to shut down and reboot your computer because some system files have been added. Rebooting the computer will load these new files to allow the Developer API Extensions system to work properly.

Chapter 3. Howdy, World!

In this chapter we will step through the process of migrating a simple Windows 32-bit application to an OS/2 32-bit application using Developer API Extensions. The application analyzed in this chapter should be quite familiar to all programmers; it simply displays "Howdy!" in its client window.

After we have looked at the basic steps involved in migrating an application, we will discuss the migration of programs with additional functions, including:

- Menu bar
- Keyboard accelerators
- Dialog boxes

In 3.2, “Enhancing Your Application” on page 92, we will add each of these common window controls to the Howdy application and show any additional steps needed to use them in an Open32 application.

3.1 Overview of the Migration Process

Figure 83 on page 80 shows the files used in both Win32 and Open32 application development. The primary C source code and header file are shared between the two platforms, while the resources and other files are platform dependent. In this chapter we will begin with a completely functioning Win32 application and modify it so that it can be compiled for either Windows or OS/2.

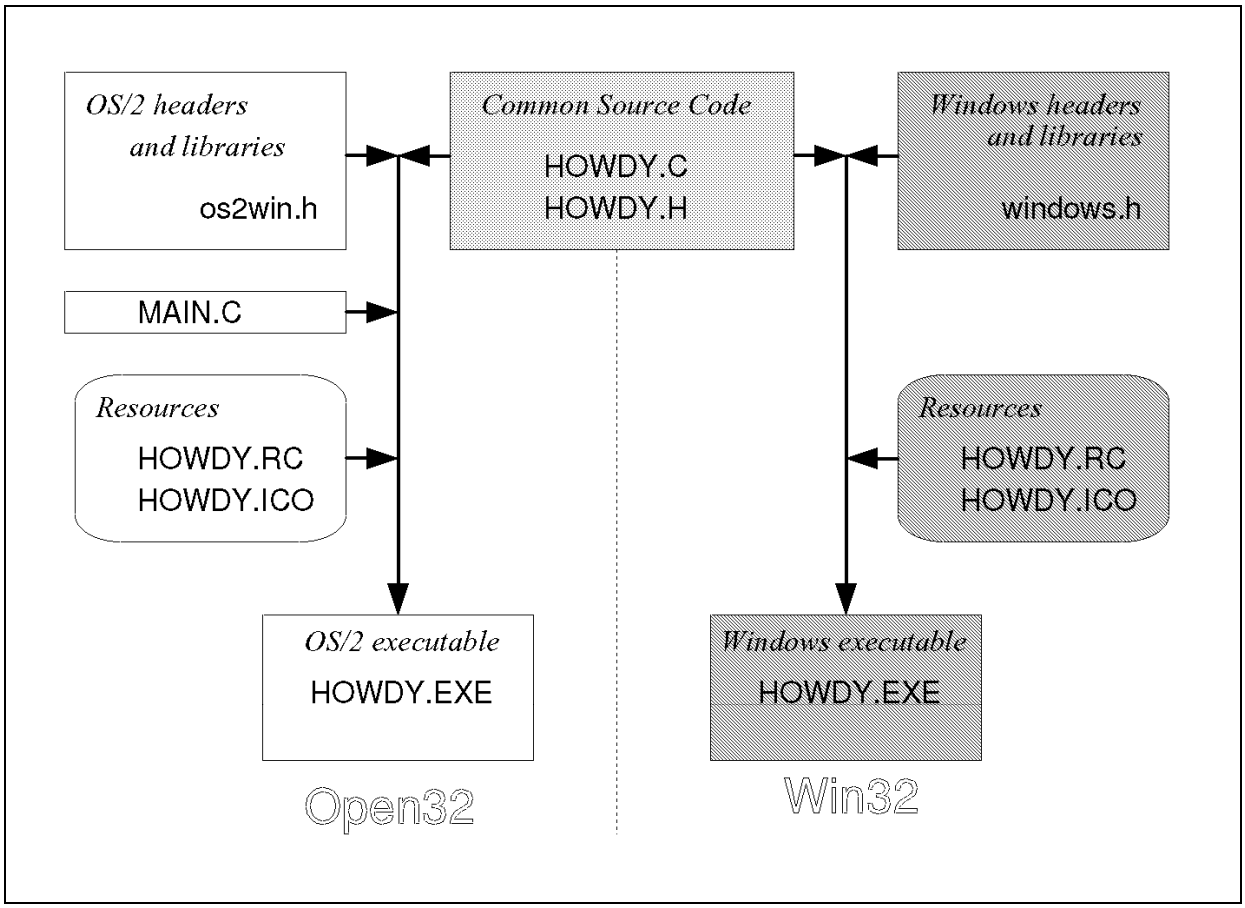


Figure 83. Structure of Howdy Source Files for Both Platforms

The migration process consists mostly of converting the source files from their original Win32 format to OS/2 format. Some files require only a few simple changes, while others must be processed by automated conversion programs. The general order recommended for converting the files is:

1. Copy the all source files to a new location.
2. Change all <windows.h> references to <os2win.h>.
3. Recompile C/C++ source code.
4. Convert Resource Compiler files.
5. Convert graphical resources (such as icons and bitmaps).
6. Compile resources.
7. Compile MAIN.C.
8. Create a new DEF file.
9. Link the application and bind the resources.

10. Run and test the new OS/2 application.

This order is recommended but is not required. You will most likely create a makefile and use NMAKE to automate the compilation process and possibly the migration process. In this chapter, a makefile is not used to show you the individual programs used to migrate and compile the Howdy application. A makefile is provided on the CD-ROM for both Windows and OS/2.

3.1.1 Copying the Source Files

The migration process is best done by replacing Win32 format files with OS/2 format files with the same name. Because this will erase or rename the Win32 format files in the working directory, we recommend that you copy the files to a separate directory before migrating. For the Howdy application, the original Win32 source code is in HOWDY BASE WIN32 on the CD-ROM. To step through the migration process on your own computer, copy the files to your hard drive. The recommended location is OPEN32 HOWDY BASE MIGRATE on any drive. See Figure 84 on page 82 for an example.

```
[D:\]MD OPEN32

[D:\]MD OPEN32\HOWDY

[D:\]MD OPEN32\HOWDY\BASE

[D:\]MD OPEN32\HOWDY\BASE\MIGRATE

[D:\]CD OPEN32\HOWDY\BASE\MIGRATE

[D:\OPEN32\HOWDY\BASE\MIGRATE]COPY  F:\HOWDY\BASE\WIN32\*
F:\HOWDY\BASE\WIN32\HOWDY.MAK
F:\HOWDY\BASE\WIN32\HOWDY.RC
F:\HOWDY\BASE\WIN32\HOWDY.ICO
F:\HOWDY\BASE\WIN32\HOWDY.EXE
F:\HOWDY\BASE\WIN32\HOWDY.C
F:\HOWDY&Bsl.BASE\WIN32\HOWDY.H
       6 file(s) copied.

[D:\OPEN32\HOWDY\BASE\MIGRATE]
```

Figure 84. Copy the Source Files from the CD-ROM to your Hard Drive

3.1.2 Changing the Source Code

For the Howdy program, only one small change is required in the source code. The one item that needs to be changed is the name of the included header file in HOWDY.C. Normally, Win32 applications include WINDOWS.H, but to compile as a Developer API Extensions application they must instead include OS2WIN.H. You can comment out the original include statement and replace it, as shown in Figure 85. In 3.2, “Enhancing Your Application” on page 92, we will look at a way of letting the compiler choose the correct include file.

```
//#include <windows.h>
#include <os2win.h>
```

Figure 85. Changes to HOWDY.C

3.1.3 Recompiling the Source Code

After the header statement has been changed for OS/2 as described in 3.1.2, “Changing the Source Code” on page 82, you are ready to recompile the source code. Figure 86 shows how to invoke the VisualAge C++ compiler from the OS/2 command prompt with the recommended options. An explanation of the compiler options is given in Table 2.

Table 2. VisualAge C++ Compiler Options

Option	Meaning
/c	Compile the source file into an object (OBJ) file, but do not invoke the linker.
/Ss	Allow the double-slash (//) to be used as a one-line comment.

```
[D:\OPEN32\HOWDY\BASE\MIGRATE]icc /c /Ss howdy.c
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

[D:\OPEN32\HOWDY\BASE\MIGRATE]
```

Figure 86. Recompiling HOWDY.C

3.1.4 Converting Resource Compiler Files

You also use the SMART tool to convert your Win32 Resource Compiler files to OS/2 format. Select **Translate Resources** from the SMART menu, as shown in Figure 87 on page 84.

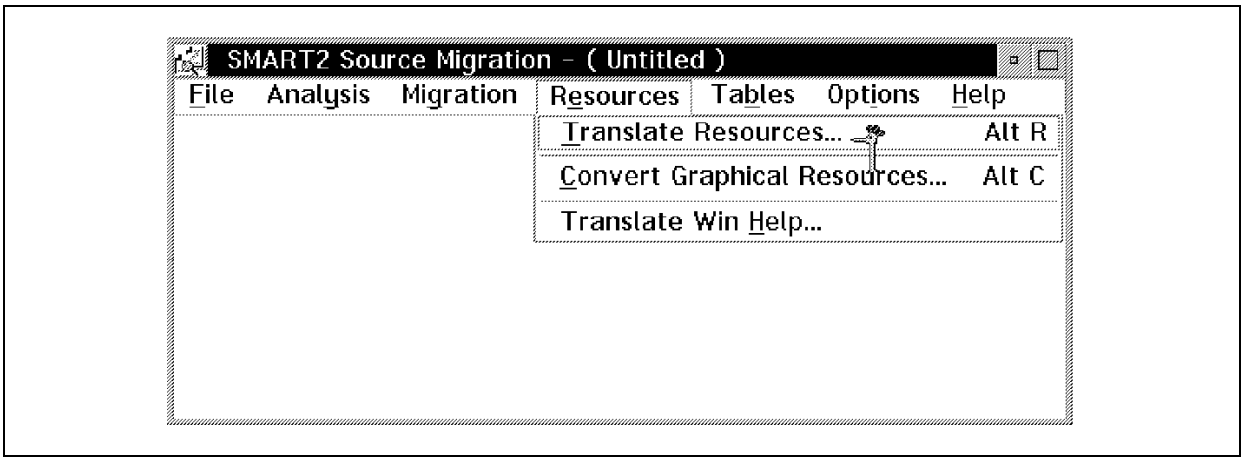


Figure 87. Selecting Translate Resources

SMART will open the Resource Translation dialog box, shown in Figure 88 on page 85. Select the name of the Resource Compiler file to be translated at the top of the dialog box. If you used String IDs in the resource file, you will need to check the box next to Support String ID.

You also have the option of using a Mapping Mask to rename files as they are processed. Since we are migrating the application in place, we select **Overwrite Original**. SMART will the save the old Win32 file under the name HOWDY.RCX and will name the new OS/2 file HOWDY.RC.

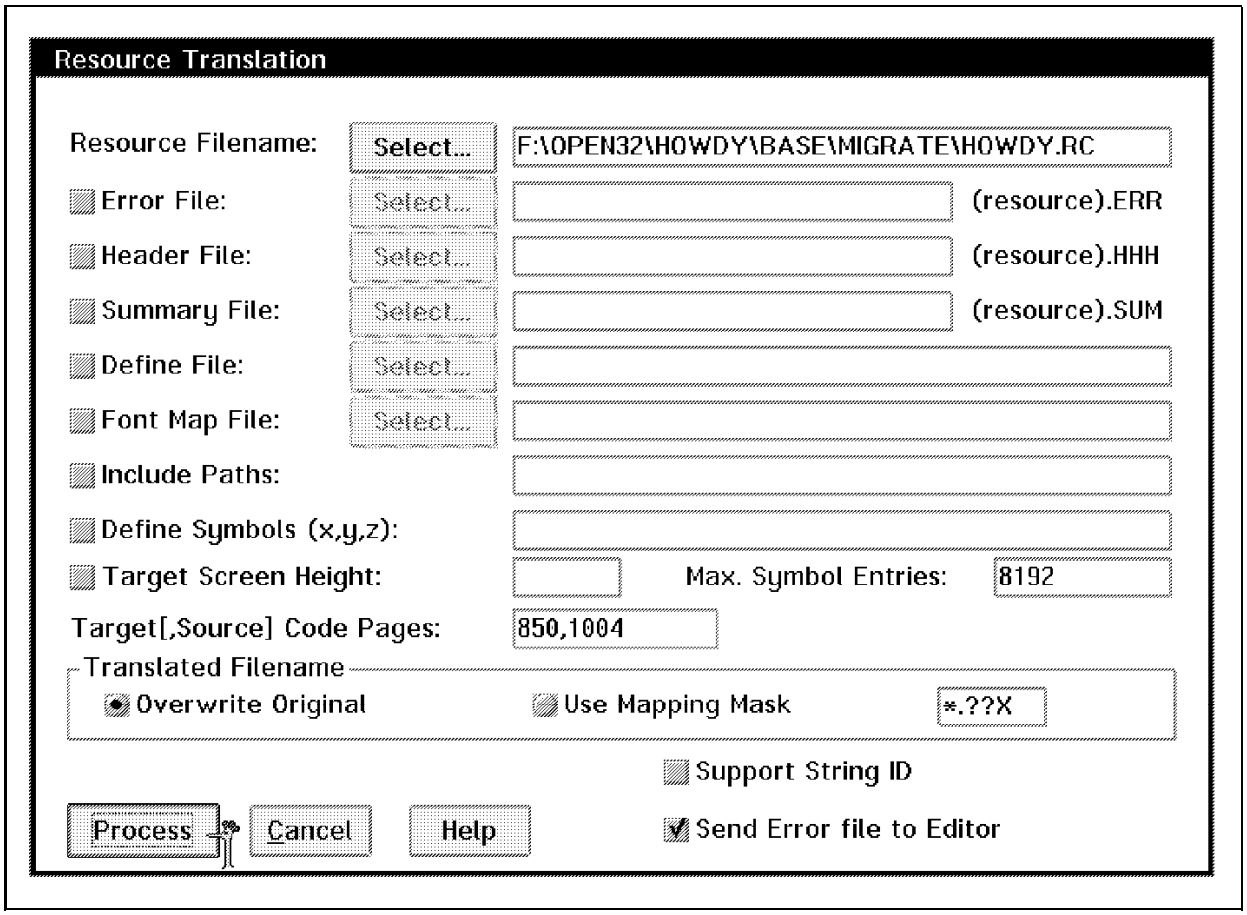


Figure 88. Resource Translation Dialog Box

After SMART finishes processing the Resource Compiler file, it will show you the results. Your results should look similar to those shown in Figure 89.

```
SMART Windows to OS/2 Resource Translation Version 1.00

Copyright (c) 1993, 1994 One Up Corporation


Command line:
g:\SMART\SMARTRC.EXE   -G 850,1004 -R  -X *.??X -E F:\OPEN32\HOWDY\BASE\MIGRATE\HOWDY.ERR -S
F:\OPEN32\HOWDY\BASE\MIGRATE\HOWDY.SUM F:\OPEN32\HOWDY\BASE\MIGRATE\HOWDY.RC

I1003 Converting from code page <1004> to code page <850>
```

Figure 89. Results of the Resource Compiler Translation

3.1.5 Converting the Resources

The next step in the conversion process is to convert Win32 Resource Compiler files to OS/2 format. The SMART tool can be used to automate this process. Figure 90 shows the main SMART tool window with the Resources menu displayed.

If you select **Convert Graphical Resources** from the Resources pull-down menu shown in Figure 90, then the Graphical Resources dialog will be displayed, as shown in Figure 91 on page 87.

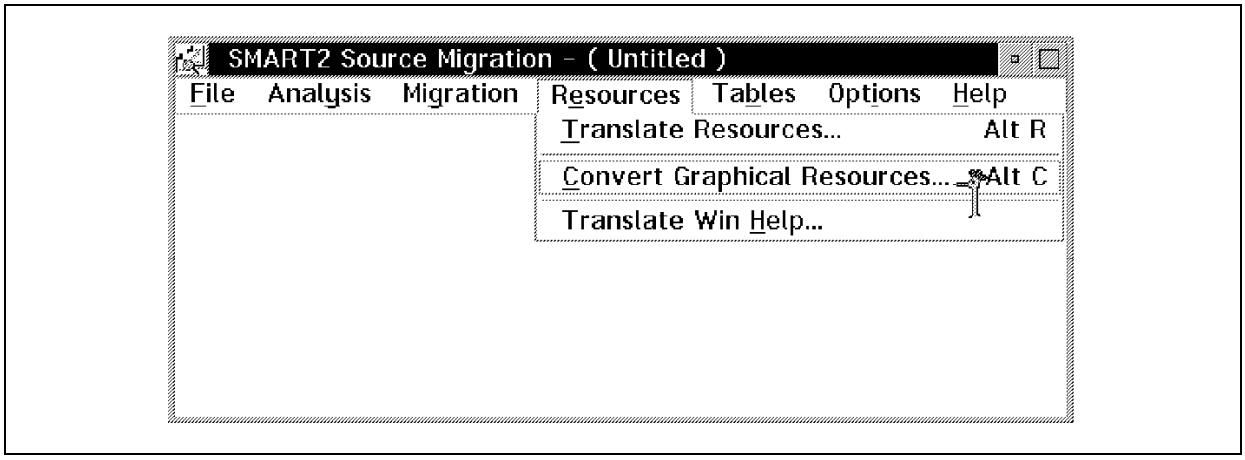


Figure 90. Selecting Convert Graphical Resources in SMART

For Howdy, the only graphical resource we need to convert is the icon, HOWDY.ICO. Since you have already copied the files to a new directory and will be migrating the copied files, you should let SMART overwrite the old Win32 icon with the new OS/2 format icon. You can either type the name of the icon with the full directory path into the edit box, or press the **Select** button to interactively select the file to be converted.

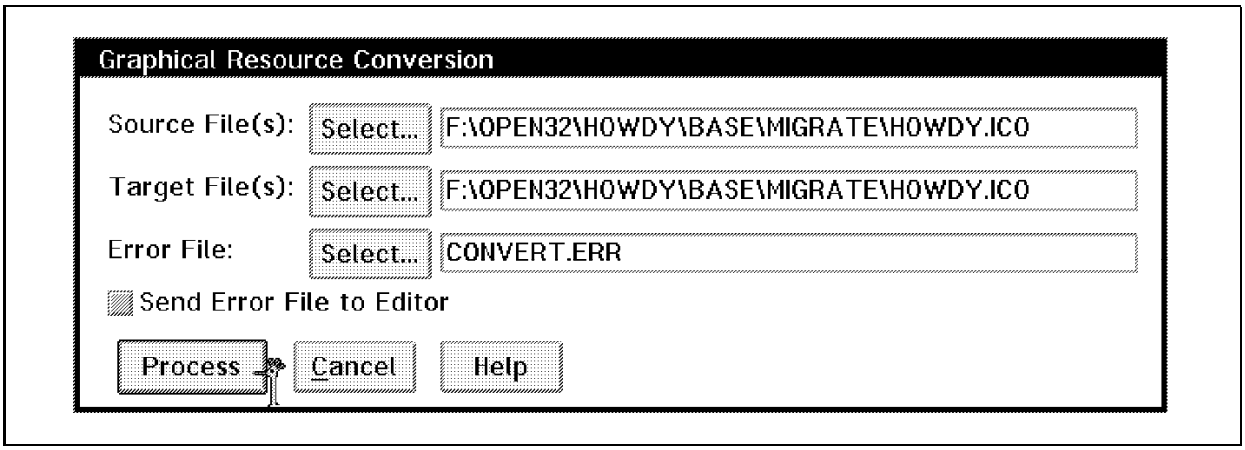


Figure 91. Selecting Files to Convert

After you have filled in the source and target files as shown in Figure 91, you can press the **Process** button and SMART will convert the icon.

SMART will capture the output of the conversion process and will create a report like the one shown in Figure 92.

```
SMART Windows to OS/2 Graphical Resource File Conversion Version 1.11

Copyright (c) 1994, 1995 One Up Corporation


I1101 Converting Windows icon <G:\SMART\SMTCVT.XXX> to OS/2 icon
  Converted file saved as <F:\OPEN32\HOWDY\BASE\MIGRATE\HOWDY.ICO>

  Files Processed:     1
    Files Converted:     1
      Warnings:     0    Errors:     0
```

Figure 92. Results of the Icon Conversion

3.1.6 Recompiling the Resource Compiler file

Now that you have converted the Resource Compiler file and the graphical resources, you can recompile them for OS/2. Enter:

```
rc /r howdy.rc
```

at the command line and the Resource Compiler will process the file and produce a compiled binary named HOWDY.RES. The `/r` option tells the Resource Compiler to create the binary RES file instead of binding the

resources directly to an executable file. The results of the command are shown in Figure 93 on page 88.

```
[F:\OPEN32\HOWDY\BASE\MIGRATE]rc /r howdy.rc
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Creating binary resource file howdy.RES
RC:  RCPP -E -D RC_INVOKED -W4 -f howdy.rc -ef H:\IBMCPP\BIN\RCPP.ERR -I H:\IBMC
PP\INCLUDE -I H:\IBMCPP\INCLUDE\OS2 -I H:\IBMCPP\INC -I H:\IBMCPP\INCLUDE\SOM -I
 H:\TOOLKIT\BETA\H -I H:\TOOLKIT\SOM\INCLUDE -I H:\TOOLKIT\H -I H:\TOOLKIT\INC -
 I F:\TOOLKIT\BETA\H -I F:\TOOLKIT\SOM\INCLUDE -I F:\TOOLKIT\H -I . -I F:\TOOLKIT
 \INC

howdy.rc.

[F:\OPEN32\HOWDY\BASE\MIGRATE]
```

Figure 93. Results of Resource Compiler

3.1.7 Compiling MAIN.C

You will recall that all Win32 applications have as their application entrypoint a WinMain() function. OS/2 applications do not have a WinMain() function; their entrypoint is the standard C/C++ function main(). For this reason, Developer API Extensions applications must compile an extra file which contains a main() function to call the Win32 WinMain() function. IBM Developer's Toolkit for OS/2 Warp ships with MAIN.C, which contains the needed main() function to call your Win32 application's WinMain(). You need to copy the file from its initial location, \TOOLKIT\SAMPLES\DAPIE\WINMAIN\MAIN.C, to the directory where you are migrating your application. A copy of MAIN.C from the toolkit is shown in Figure 94 on page 89.

```
/*****************************************************************************
 * Copyright:                                                                *
 *   Licensed Materials - Property of IBM                                     *
 *   (C) Copyright IBM Corp. 1995                                             *
 *   All Rights Reserved                                                      *
 *****************************************************************************
 * File: main.c                                                              *
 *****************************************************************************
 * Description:                                                              *
 *   Sample "main" wrapper for applications/executables.                      *
 *****************************************************************************/
#include <os2win.h>

int main(int argc, char *argv[], char *envp[])
{
   /* Call WinCallWinMain to start the application.
    */
   return WinCallWinMain( argc, argv, &WinMain, SW_SHOWNORMAL );
}
```

Figure 94. MAIN.C from the OS/2 Warp Toolkit

Once you have a copy of MAIN.C, you must compile it. Both these steps are shown in Figure 95.

```
[F:\OPEN32\HOWDY\BASE\MIGRATE]copy h:\toolkit\samples\dap ie\winmain\main.c
      1 file(s) copied.

[F:\OPEN32\HOWDY\BASE\MIGRATE]icc /c /Ss main.c
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

[F:\OPEN32\HOWDY\BASE\MIGRATE]
```

Figure 95. Copy and Compile MAIN.C

3.1.8 Creating a New DEF File

A link definitions file is necessary to link any OS/2 application, so a new DEF file must be written for the link step in developing or migrating Developer API Extensions applications. The most important lines tell the linker the title of the application and the stacksize of the executable.

The DEF file needed to compile HOWDY is shown in Figure 96 on page 90.

```
NAME       HOWDY WINDOWAPI
DESCRIPTION 'Howdy, World! Sample Application (C) IBM, 1996'
STACKSIZE   65536
```

Figure 96. HOWDY.DEF

Open32 applications require relatively large stacks because of the way such applications work under OS/2. The Developer API Extensions application calls OS/2 functions which correspond to Win32 functions, and they in turn call other OS/2 functions to perform the necessary actions. This quickly creates very large function call trees, which must be held on the stack. As a result, Developer API Extensions applications require large stacks. The recommended minimum is 65536 bytes.

For more information on the DEF file format, see the IBM Developer's Toolkit for OS/2 Warp.

3.1.9 Linking the Application and Binding the Resources

The final step is to link the compiled source code into an executable file and bind the resources to it. At the command line, link the files with:

```
ilink howdy.obj main.obj pmwinx.lib howdy.def
```

IBM's new linker, `ILINK`, automatically categorizes files based on their extensions, so you do not need to specify which files are the object files, which are libraries, and so on.

The extra library linked with the application, `PMWINX.LIB`, contains the necessary code to link any Open32 application to the run-time Developer API Extensions DLLs in OS/2. You must manually specify it for any Open32 application.

After the link step, bind the resources to the executable with the Resource Compiler. At the command line, type:

```
rc howdy.res howdy.exe
```

The Resource Compiler will add the application's resources to the executable file.

The results of these last two steps are shown in Figure 97 on page 91.

```
[F:\OPEN32\HOWDY\MIGRATE]ilink howdy.obj main.obj pmwinx. lib howdy.def

IBM(R) Linker for OS/2(R), Version 01.00.05
(C) Copyright IBM Corporation 1988, 1995.
(C) Copyright Microsoft Corp. 1988, 1989.
 - Licensed Material - Program-Property of IBM - All Rights Reserved.


[F:\OPEN32\HOWDY\MIGRATE]rc howdy.res howdy.exe
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Reading binary resource file howdy.res

.
Writing resources to OS/2 v2.0 Linear .EXE file
Writing 1 DEMAND resource object(s)
  Writing:  876 bytes in 1 page(s)
     101.1 (874 bytes)


[F:\OPEN32\HOWDY\BASE\MIGRATE]
```

Figure 97. Link and Bind Application

3.1.10 Testing the Application

Now, you should have a newly compiled HOWDY application for OS/2. When you run the program, the main window should appear, as shown in Figure 98 on page 92.

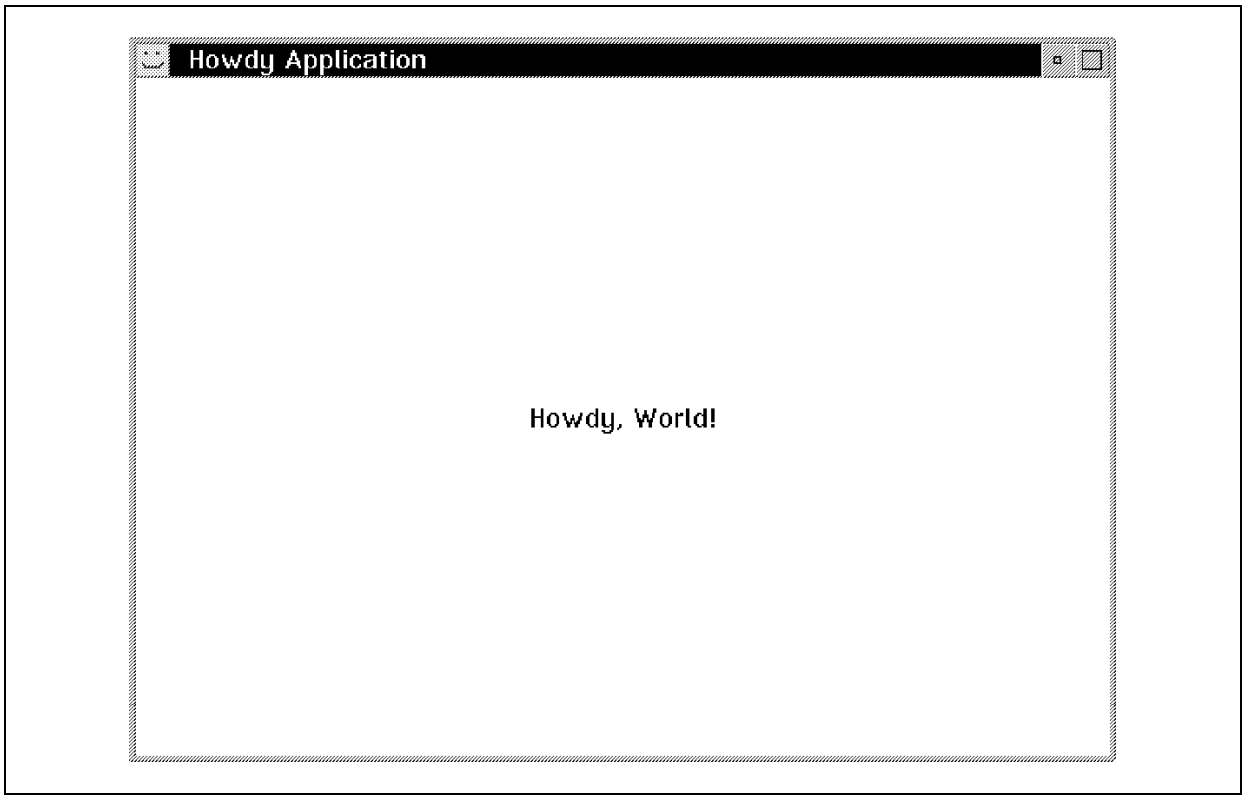


Figure 98. Your New OS/2 Application

3.2 Enhancing Your Application

The Howdy application is fine as a stand-alone programming example, but it does not demonstrate any real-world programming situation. In this section, you will add the following standard Win32 resources to the application:

- Menu (See Section 3.2.1, “Adding a Menu”)
- Accelerator (See Section 3.2.2, “Adding Accelerators” on page 95)
- Dialog box (See Section 3.2.3, “Adding Dialog Boxes” on page 96)

3.2.1 Adding a Menu

Almost every Win32 application has a menu bar above its client window, so it is a logical choice as the first resource to add to the Howdy application.

The Win32 source files for the menu-enabled Howdy are on the CD in the directory HOWDY MENU WIN32. The common files, shared between both Win32 and OS/2 versions of Howdy, are in HOWDY MENU. The OS/2,

specific source files are in HOWDY MENU OS2. This is the structure used for the sample applications throughout the remainder of this redbook.

3.2.1.1 Copying the Source Code

We recommend copying the source code for the Howdy application to your hard drive in the directory OPEN32 HOWDY MENU. You can then copy the Win32 source files to a MIGRATE directory. Both of these steps are shown in Figure 99.

```
[D:\OPEN32\HOWDY]md menu

[D:\OPEN32\HOWDY]cd menu

[D:\OPEN32\HOWDY\MENU]copy f:\howdy\menu\*
F:\howdy\menu\HOWDY.C
F:\howdy\menu\HOWDY.H
       2 file(s) copied.

[D:\OPEN32\HOWDY\MENU]md migrate

[D:\OPEN32\HOWDY\MENU]cd migrate

[D:\OPEN32\HOWDY\MENU\MIGRATE]copy f:\howdy\menu\win32\*
F:\howdy\menu\win32\HOWDY.EXE
F:\howdy\menu\win32\HOWDY.ICO
F:\howdy\menu\win32\HOWDY.MAK
F:\howdy\menu\win32\HOWDY.OBJ
F:\howdy\menu\win32\HOWDY.RC
F:\howdy\menu\win32\MAKEFILE
       6 file(s) copied.

[D:\OPEN32\HOWDY\MENU\MIGRATE]
```

Figure 99. Copying the Howdy Menu Source Files

The HOWDY.C and HOWDY.H source files are shared between both the OS/2 and the Win32 version of Howdy. This is possible because the source code uses a precompiler `#ifdef` statement to let the compiler decide whether to include `<windows.h>` or `<os2win.h>`. This statement is shown in Figure 100 on page 94. This technique is used for most applications in this redbook.

```
#ifdef OS2                  // Compiling for OS/2
#include <os2win.h>
#else                      // Compiling for Windows
#include <windows.h>
#endif
```

Figure 100. HOWDY.C: Precompiler Statement

The process of migrating an application with a menu bar is essentially identical to the process used in 3.1, “Overview of the Migration Process” on page 79. The only difference is when you use SMART to translate the resources, SMART's output will be displayed as shown in Figure 101.

```
SMART Windows to OS/2 Resource Translation Version 1.00

Copyright (c) 1993, 1994 One Up Corporation


Command line:
g:\SMART\SMARTRC.EXE   -G 850,1004 -R  -X *.??X -E
F:\OPEN32\HOWDY\MENU\MIGRATE\HOWDY.ERR -S
F:\OPEN32\HOWDY\MENU\MIGRATE\HOWDY.SUM
F:\OPEN32\HOWDY\MENU\MIGRATE\HOWDY.RC

I1003 Converting from code page <1004> to code page <850>

W2013 Macro redefinition : <IDR_MENU1>
W2013 Macro redefinition : <MM_FILEEXIT>
F:\OPEN32\HOWDY\MENU\MIGRATE\HOWDY.RC(57:11) : W2006 Submenu item <"&File">
assigned identifier of <0xF200>
```

Figure 101. SMART Output when Translating a Menu

The messages about:

`Macro redefinition`

simply mean that SMART is translating menu item definitions from Win32 format to OS/2 format. The last message, in which `<"&File">`is assigned a hexadecimal identifier, tells you that SMART is automatically assigning the submenu a required number. The identifier is required by the Resource Compiler, and is available to native OS/2 applications. Because you are migrating a Win32 application using Developer API Extensions, the identifier is invisible to your application and can be disregarded.

Now that the resources are ready, you can finish migrating the application using the same steps as in the previous section. This will create an application as shown in Figure 102 on page 95.

Note

The C source code for Howdy with a menu differs from the base C source used in 3.1, “Overview of the Migration Process” on page 79. The menu application requires extra code to respond to the menu items.

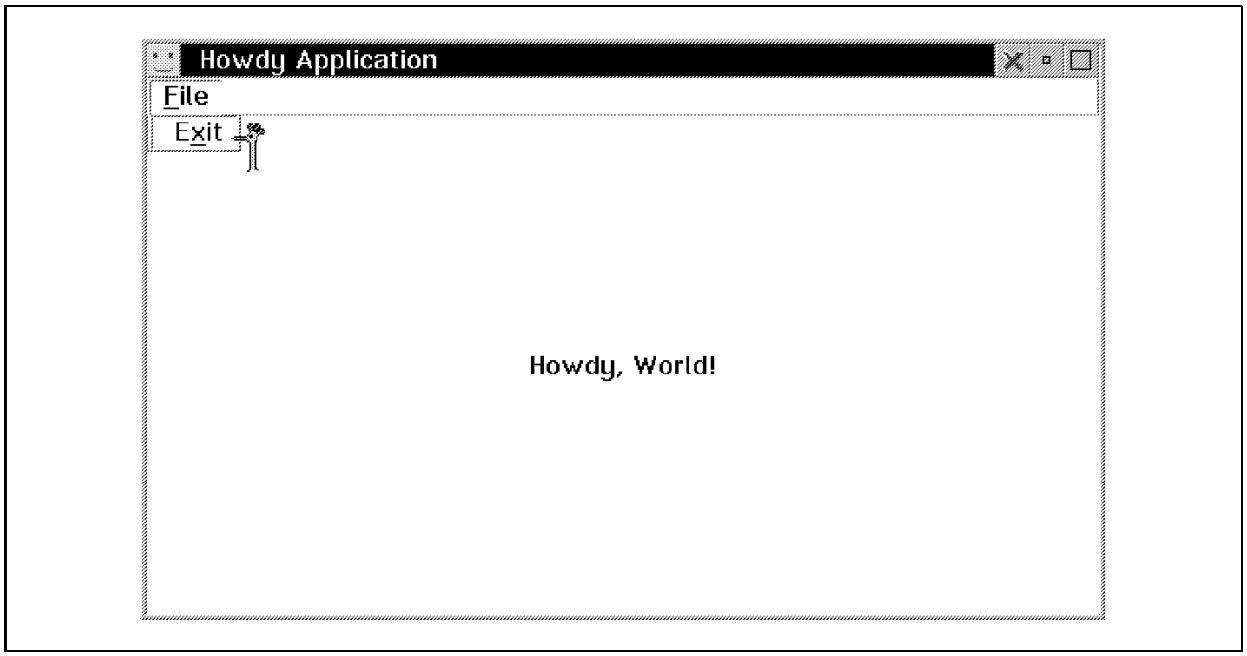


Figure 102. Howdy with a Menu

If you have problems, refer to 3.1, “Overview of the Migration Process” on page 79 and Appendix A, “Common Problems and Easy Solutions” on page 255 for solutions.

3.2.2 Adding Accelerators

Once you have a menu, it is logical that you would want to add keyboard accelerators for some of the functions. SMART will translate the resource file definitions for you automatically.

After SMART finishes processing, you will get a message like the one shown in Figure 103 on page 96.

```
SMART Windows to OS/2 Resource Translation Version 1.00

Copyright (c) 1993, 1994 One Up Corporation


Command line:
g:\SMART\SMARTRC.EXE   -G 850,1004 -R  -X *.??X -E F:\OPEN32\HOWDY\ACCEL\MIGRATE\HOWDY.ERR
-S F:\OPEN32\HOWDY\ACCEL\MIGRATE\HOWDY.SUM F:\OPEN32\HOWDY\ACCEL\MIGRATE\HOWDY.RC

I1003 Converting from code page <1004> to code page <850>

F:\OPEN32\HOWDY\ACCEL\MIGRATE\HOWDY.RC(4:1) : W2017 Include file name <windows.h> changed to
<os2.h>
W2013 Macro redefinition : <IDR_MENU1>
W2013 Macro redefinition : <IDR_ACCELERATOR1>
W2013 Macro redefinition : <MM_FILEEXIT>
F:\OPEN32\HOWDY\ACCEL\MIGRATE\HOWDY.RC(66:11) : W2006 Submenu item <"&File"> assigned
identifier of <0xF200>
```

Figure 103. SMART Results with Accelerators

Note, the statement:

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

was not changed to

```
#include <os2win.h>
```

This is due to the nature of Developer API Extensions. The resources you bind to your executable file are in native OS/2 format, so the include file for resources must be the standard OS/2 include file. Note, the end result is your accelerator table is loaded and used as a native OS/2 accelerator table, although the functions you call with it appear to be Win32 functions.

Note

If you ever find that your accelerator table does not work properly, check your resource file to see if <os2win.h> has been included by mistake. The Resource Compiler will not produce any build-time errors if this happens. However, due to differences in virtual key definitions between Win32 and OS/2, the keys will not work as they should.

3.2.3 Adding Dialog Boxes

Dialog boxes are also very common in Win32 programs. Again, SMART will do most of the work for you in converting the dialog box to OS/2. The output of the SMART conversion is shown in Figure 104 on page 97.

```
SMART Windows to OS/2 Resource Translation Version 1.00

Copyright (c) 1993, 1994 One Up Corporation


Command line:
g:\SMART\SMARTRC.EXE   -G 850,1004 -R  -X *.??X -E F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.ERR
-S F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.SUM F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC

I1003 Converting from code page <1004> to code page <850>

F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(4:1) : W2017 Include file name <windows.h>
changed to <os2.h>
W2013 Macro redefinition : <IDR_MENU1>
W2013 Macro redefinition : <IDR_ACCELERATOR1>
W2013 Macro redefinition : <IDD_SETTINGS>
W2013 Macro redefinition : <IDC_SHOWICON>
W2013 Macro redefinition : <IDC_COLORBLACK>
W2013 Macro redefinition : <IDC_COLORRED>
W2013 Macro redefinition : <IDC_COLORGREEN>
W2013 Macro redefinition : <IDC_CHECKMESSAGE>
W2013 Macro redefinition : <IDC_MESSAGE>
W2013 Macro redefinition : <MM_FILEEXIT>
W2013 Macro redefinition : <MM_DIALOGSETTINGS>
W2013 Macro redefinition : <IDC_STATIC                    -1>
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(66:11) : W2006 Submenu item <"&File"> assigned
identifier of <0xF200>
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(70:11) : W2006 Submenu item <"&Dialogs"> assigned
identifier of <0xF201>
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(95:23) : E3033 <DS_CENTER> is not a recognized
keyword
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(97:6) : W2001 Font <MS Sans Serif> mapped
to <Helv>
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(108:26) : W2016 Control identifier <IDOK>
changed to <DID_OK>
F:\OPEN32\HOWDY\DIALOG\MIGRATE\HOWDY.RC(109:30) : W2016 Control identifier <IDCANCEL>
changed to <DID_CANCEL>
```

Figure 104. Results of Dialog Box Migration

You will notice in Figure 104 that there was one error:

```
HOWDY.RC(95:23) : E3033 <DS_CENTER> is not a recognized keyword
```

The DS_CENTER keyword instructs Windows to put the dialog box in the center of the screen. There is not an equivalent keyword in OS/2, but you still have some control over dialog box location. You should finish compiling Howdy to make sure it is at least functional.

You will probably encounter one small error when you try to recompile HOWDY.C under OS/2, as shown in Figure 105 on page 98.

```
howdy.c(42:4) : error EDC3013: Identifier "strcpy" is undefined.
NMAKE : fatal error U1077: 'G:\OS2\CMD.EXE' : return code '12'
Stop.
```

Figure 105. Howdy Compiler Error

Although `strcpy()` is a standard C function, the VisualAge C++ compiler needs its definition. The function is defined in the header file `string.h`, so you need to add:

```
#include <string.h>
```

to `HOWDY.C`. If you are not sure what file a function is defined in, you can look in the "C Library Reference" in the "VisualAge C++ Information" folder.

With `strcpy()` defined correctly, the Howdy application will build normally.

Now you can run the application and test the dialog box. Selecting **Settings** from the menu will activate the dialog, shown in Figure 106. Pressing **Ctrl-S** in the main window should activate the dialog box, but depending on your version of SMART, it may not. Some versions are known to have problems translating accelerator tables. If Ctrl-S does not work for you, you need to manually edit HOWDY.RC and change the accelerator definition by changing the letter *S* to a lowercase *s* and removing the SHIFT keyword. For more information about accelerator translation problems, see 3.2.4, "Resource Differences and SMART Limitations" on page 100.

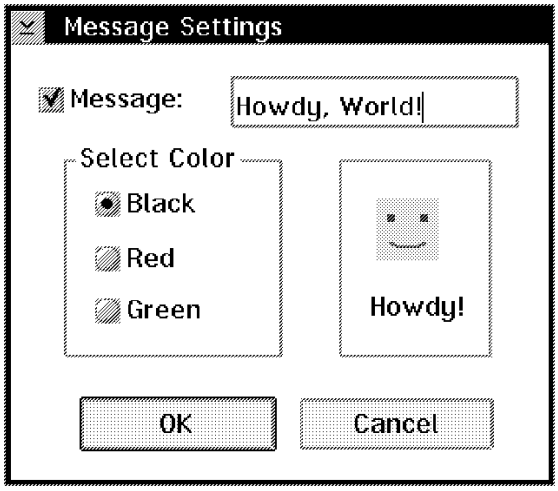


Figure 106. Message Settings Dialog Box

You can modify the text and color of Howdy's message in the dialog box. You can also disable the message completely by removing the checkmark from the Message checkbox.

Now that you know that Howdy works properly, you can modify it so the dialog box appears in a more useful location. You will find it most helpful to align dialog box locations with respect to your application window rather than to the screen. Screen size changes from system to system and can vary greatly, but your application window will usually be about the same size. By aligning the dialog to your window, you can also be sure that the dialog appears near or over your application window.

To make the dialog box position relative to your application window, delete the FS_SCREENALIGN keyword from the dialog attributes list in HOWDY.RC. This will cause the dialog box to appear in the lower left corner of your window. You can move it up and to the right by adjusting the first two numbers in the DIALOG statement. Figure 107 shows these modifications to the dialog box definition.

```
DLGTEMPLATE IDD_SETTINGS DISCARDABLE
BEGIN
  DIALOG "Message Settings", IDD_SETTINGS, 90, 75, 172, 111,
        FS_DLGBORDER,
        FCF_TITLEBAR | FCF_SYSMENU | FCF_NOMOVEWITHOWNER
  BEGIN
    CONTROL      "Message:",IDC_CHECKMESSAGE,  9,  92,  59,  10, WC_BUTTON,
                BS_AUTOCHECKBOX | WS_TABSTOP | BS_PUSHBUTTON | WS_VISIBLE
    ENTRYFIELD    "",          IDC_MESSAGE,  72,  91,  89,  11, ES_MARGIN
    CONTROL      "Black",    IDC_COLORBLACK,  27,  65,  43,  10, WC_BUTTON,
                BS_AUTORADIOBUTTON | BS_PUSHBUTTON | WS_VISIBLE
    CONTROL      "Red", IDC_COLORRED,  27,  51,  37,  10, WC_BUTTON,
                BS_AUTORADIOBUTTON | BS_PUSHBUTTON | WS_VISIBLE
    CONTROL      "Green",    IDC_COLORGREEN,  27,  38,  44,  10, WC_BUTTON,
                BS_AUTORADIOBUTTON | BS_PUSHBUTTON | WS_VISIBLE
    DEFPUSHBUTTON "OK",             DID_OK,  22,   7,  63,  14
    PUSHBUTTON   "Cancel",      DID_CANCEL,  92,   7,  63,  14
    GROUPBOX     "Select Color",IDC_COLORGROUP,17, 31,  70,  55, WS_TABSTOP | DT_MNEMONIC
    ICON         ID_HOWDYICON, IDC_SHOWICON, 117,  56,  20,  16
    LTEXT        "Howdy!",      IDC_STATIC, 115,  40,  32,   8, NOT WS_GROUP |
                SS_TEXT | DT_WORDBREAK | DT_MNEMONIC
    GROUPBOX      "",           IDC_STATIC, 105,  31,  49,  55, DT_MNEMONIC
  END
END
```

Figure 107. HOWDY.RC with Modifications

Note that remigrating the Win32 Resource Compiler file with SMART will destroy the changes you have made so that the OS/2 version of Howdy works nicely. This is especially important when you update your Win32 application and need to remigrate the new version to OS/2.

3.2.4 Resource Differences and SMART Limitations

You have already seen some of the situations that SMART cannot fully resolve. There are many problems in converting Win32 applications to OS/2 applications which Developer API Extensions and SMART cannot solve alone. You will need to take a very active role in migrating your application to OS/2.

There are a few tricks and pitfalls you should know about before continuing to the next chapter. First, there is an important difference between the way Windows and OS/2 use accelerator tables. Under Windows, the keystroke Ctrl-s is the same regardless of the state of CapsLock. Under OS/2, the keystroke Ctrl-s is different from Ctrl-S (note the difference in case), and both are different from Ctrl-Shift-S or Ctrl-Shift-s. It is therefore very important that you modify your accelerator tables after SMART conversion by duplicating accelerators that use the character keys.

For example, say you originally had the following accelerator in your Windows application:

```
"S",          MM_DIALOGSETTINGS,     VIRTKEY, CONTROL, NOINVERT
```

SMART would translate the accelerator to OS/2 format for you automatically:

```
"S",          MM_DIALOGSETTINGS, VIRTUALKEY, SHIFT, CONTROL
```

However, this is not the same accelerator key to the end user. The user must press Ctrl-Shift-s with CapsLock off to activate the accelerator. You should change the line to:

```
"s",          MM_DIALOGSETTINGS, VIRTUALKEY, CONTROL
```

This is a start, but now if the user has CapsLock on, they cannot use the accelerator. To fix this, you need to make a second accelerator for a capital ‘S’. Both are shown in Figure 108.

```
"s",          MM_DIALOGSETTINGS, VIRTUALKEY, CONTROL
"S",          MM_DIALOGSETTINGS, VIRTUALKEY, CONTROL
```

Figure 108. A Fully Functional OS/2 Accelerator Definition

You may have also noticed that the Message Settings dialog box changed slightly when it was converted to OS/2. Whereas the Win32 dialog had both the Howdy icon and the “Howdy!” text centered in the group frame, the OS/2 version was slightly different. This is due to differences in mapping coordinate systems between Win32 and OS/2, as well as differences in icon

size and text width. To realign the controls for OS/2, you can use the Dialog Editor included in the OS/2 Warp Toolkit.

Be careful when using the Dialog Editor. It will modify the compiled HOWDY.RES file, but will leave the source file HOWDY.RC unchanged. The Dialog Editor will create a separate HOWDY.DLG file which contains the modified dialog box. You can change HOWDY.RC to include HOWDY.DLG, but remember that if at some future time you again migrate the resource file from Win32 these changes will be lost and you will need to reuse the Dialog Editor and include the HOWDY.DLG in the newly migrated resource file.

While there are these small issues in migrating your application to OS/2, it is still remarkably easy to do. After these few examples, you should be fairly competent in migrating the most important parts of your application. The following chapters will deal with more complex issues which you may encounter.

Chapter 4. MDI Sample Program

This chapter describes another common source application sample which in comparison with the sample described in Chapter 3, “Howdy, World!” on page 79, has more features and functions. This sample has many of the functions commonly found in Windows' applications. The main goal of this chapter is to help you become familiar with Developer API Extensions functions and to expand on the migration steps and techniques presented in Chapter 3, “Howdy, World!” on page 79.

For OS/2 application developers who may not be familiar with the concepts of developing programs using the Windows Multiple Document Interface (MDI), this chapter will illustrate how this can be coded. Thus, Open32 provides OS/2 application developers with additional functions from which to build applications for the OS/2 environment.

4.1 Application's Overview

The sample program described in this chapter is based on Windows Multiple Document Interface (MDI). It uses MDI child windows to present four different types of information:

- Bitmap
- Graphics
- Screen capture image
- Text

Figure 109 on page 104 shows the MDI sample program with four MDI child windows open and filled with different types of information.

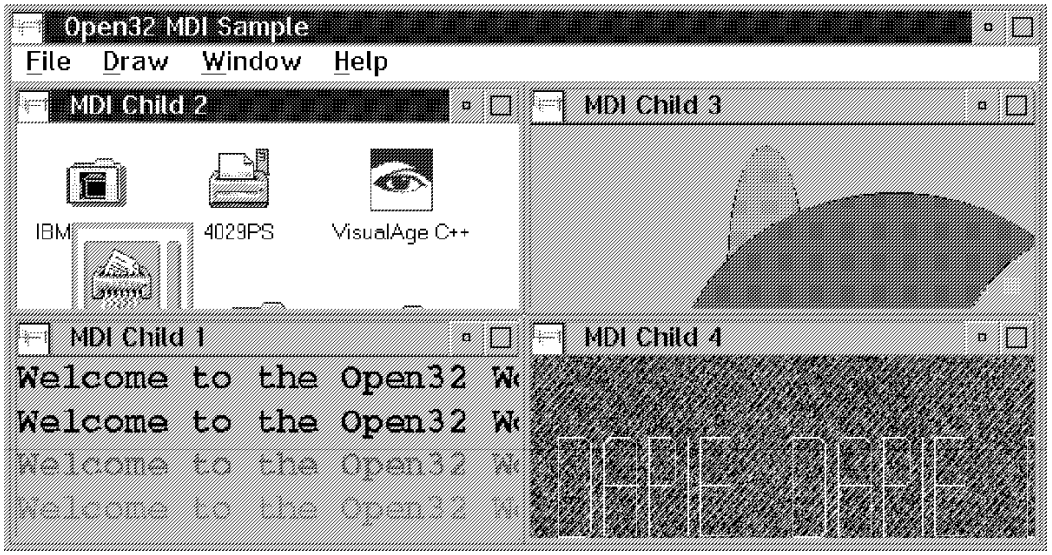


Figure 109. MDI Sample Program Overview

The MDI sample program architecture is shown in Figure 110 on page 105.

The MDI sample program starts by creating a main window, called the MDI frame window, which in turn creates a child window, called the MDI client window. Later the MDI client window creates its own child windows called MDI child windows. It is these MDI child windows which contain the information that the application presents.

There is a clean separation of responsibility among these different MDI windows in this architecture. The MDI frame window takes care of the user's interaction. The MDI client window is responsible for the creation and the management of the MDI child windows. The MDI child windows is responsible for the presentation of information in its drawing area.

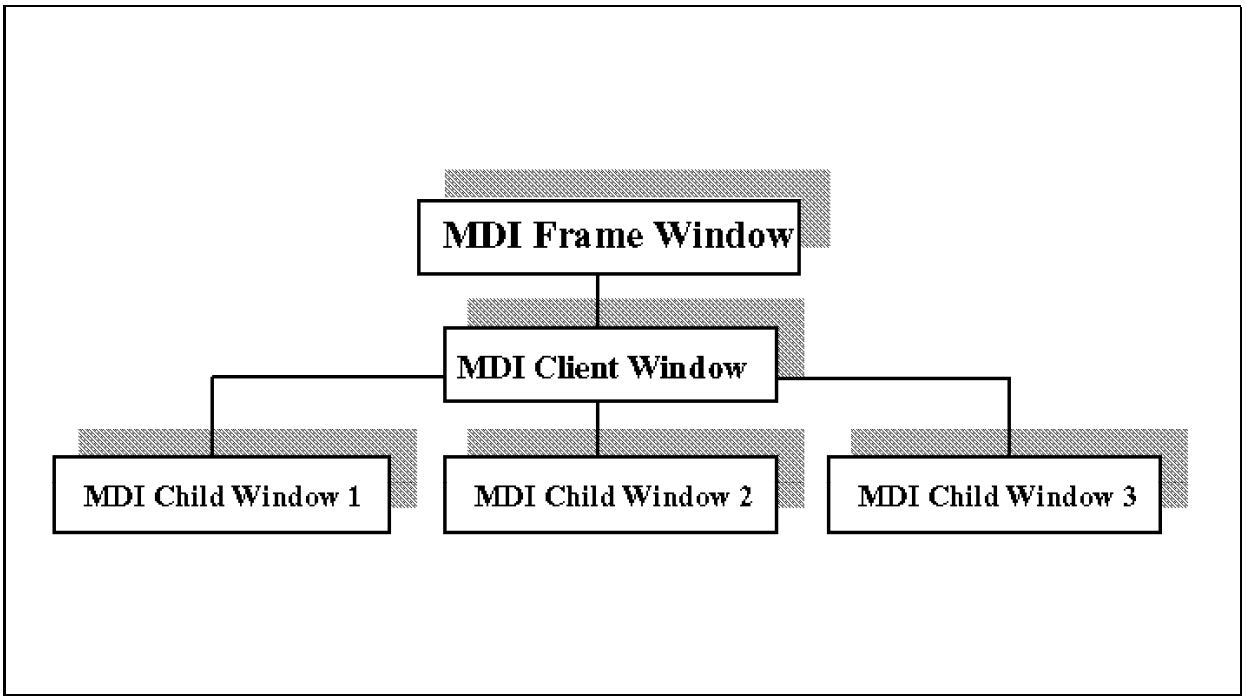


Figure 110. MDI Sample Program Architecture

The user interacts with the application through menus. When a menu command is received by the application, depending on the type of the command, the MDI frame window forwards it either to the MDI client window or to the currently active MDI child window for further processing. The Windows Multiple Document Interface provides all the means necessary for all these components to work together.

Note that OS/2 does not have built-in support for MDI. Of course programmers can code it themselves but it will not be trivial job, and it is preferable to have it as native operating system support. The Open32 support now offers this type of function to you at the operating system level for OS/2 Warp.

4.2 User's Interface

When the MDI sample program is executed, only the File and Help pull-down menus are accessible, as shown in Figure 111 on page 106. The three things you can do at this point are to create a new MDI child window (New option of File menu bar pull-down), to display the About dialog box (About option of Help menu bar pull-down) or to leave the application (Exit option of File menu bar pull-down).

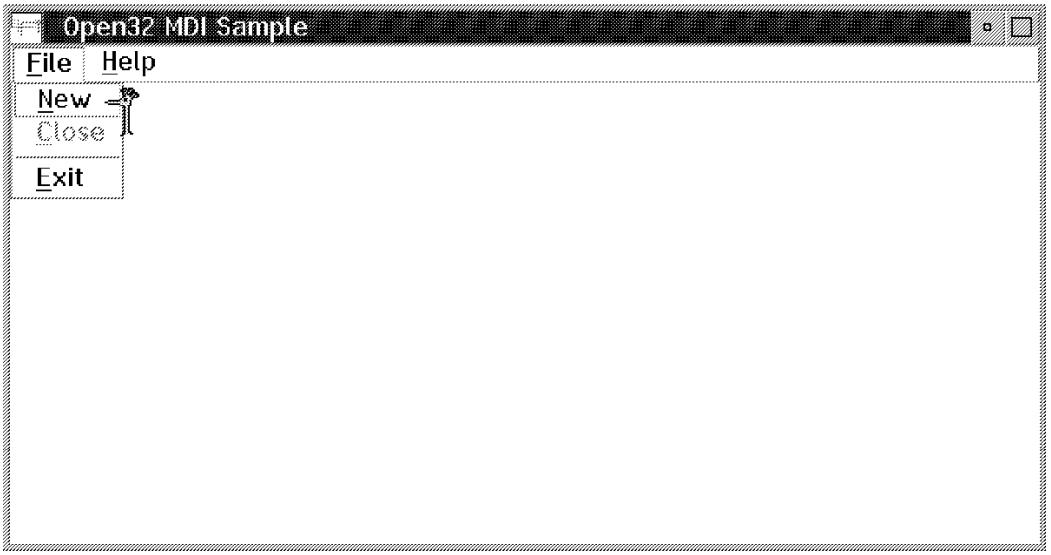


Figure 111. MDI Sample Program Main Menu

The Close option of the File menu bar pull-down is disabled, since no MDI child window has yet been open. After the creation of the first MDI child window, the Close option of the File menu bar pull-down is enabled allowing you to close the currently active MDI child window. Also two more MDI child window specific pull-down menus, Draw and Window, become available as shown in Figure 112 on page 107.

After you have opened a MDI child window, you can then fill the MDI child windows from the Draw pull-down menu by selecting Bitmap, Graphics, Screen Capture or Text option. From the Window pull-down menu, as shown in Figure 113 on page 107, you can reorganize the MDI child windows within the MDI client window using the Cascade or Tile option. Also you can arrange the icons of the MDI child windows using the Arrange Icons option. Beside these reorganization options, the Window pull-down menu displays the list of all the MDI child windows and gives you a rapid way to select one of them or to close all of them using the Close All option. You can also directly manipulate the MDI child windows through their own maximize and minimize menu system.

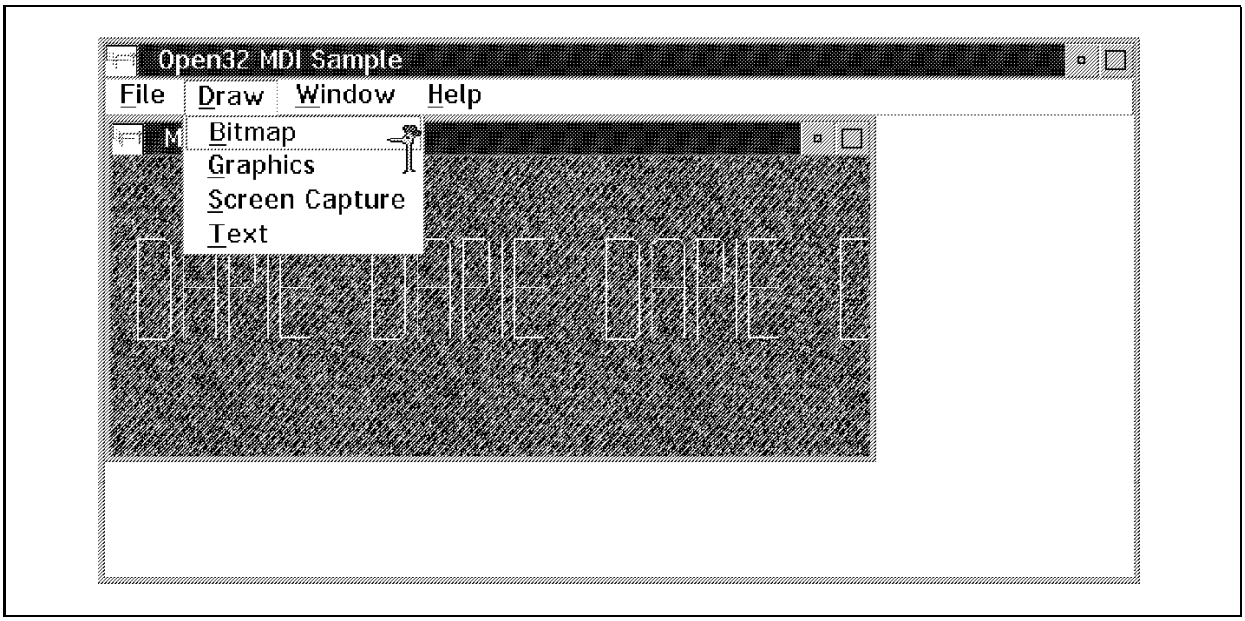


Figure 112. Draw Pull-Down Menu

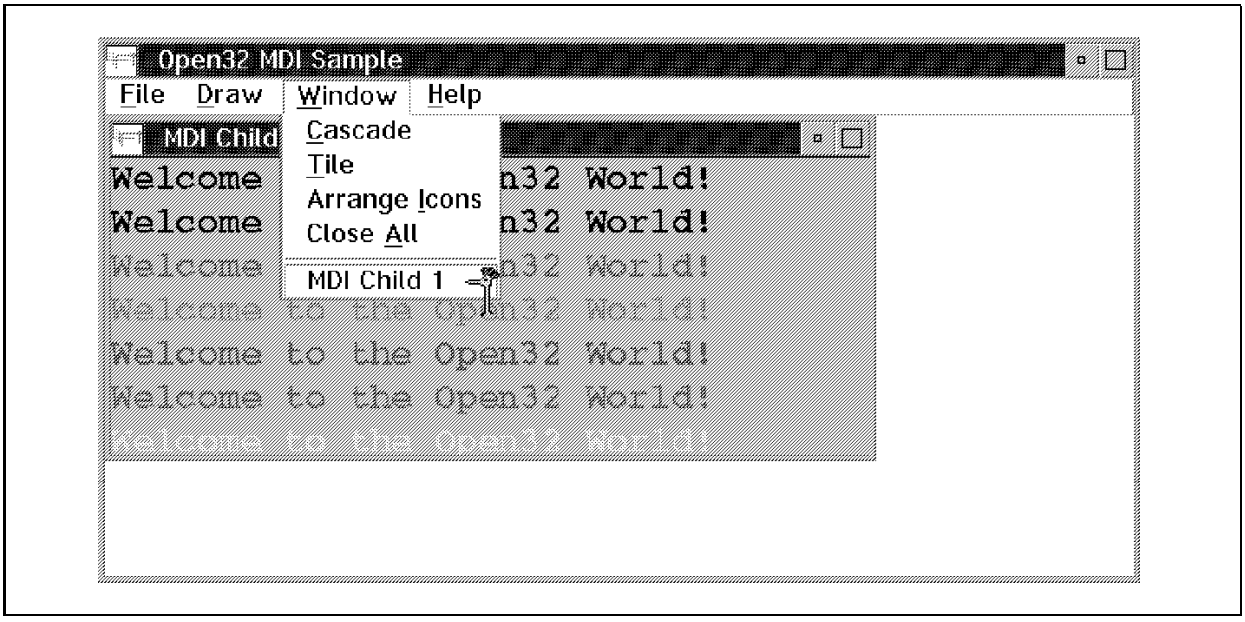


Figure 113. Window Pull-Down Menu

4.3 Summary of Win32 API Functions Used

The Win32 API functions used in the MDI sample program fall into the following categories:

- Window Management
- Graphics Device Interface
  - Graphics drawing
  - Text drawing
  - Bitmap drawing
  - Screen Capture
- Common Dialog Box

The following list groups the Win32 API functions used in the MDI sample program by the type of the object they manipulate:

- Bitmaps
  - BitBlt()
  - CreateCompatibleBitmap()
  - PatBlt()
- Device Contexts
  - CreateCompatibleDC()
  - CreateDC()
  - DeleteDC()
  - GetDC()
  - ReleaseDC()
- Drawing-Attribute and Tool
  - CreateSolidBrush()
  - DeleteObject()
  - GetObject()
  - GetStockObject()
  - SelectObject()
  - SetBkMode()
- Ellipse
  - Ellipse()

- Fonts
  - ChooseFont() (Common Dialog Box Library)
  - CreateFontIndirect()
- Menu
  - DrawMenuBar()
  - GetSubMenu()
- Message and Message Queues
  - DispatchMessage()
  - GetMessage()
  - PostQuitMessage()
  - SendMessage()
  - TranslateMessage()
- Multiple Document Interface
  - DefFrameProc()
  - DefMDIChildProc()
- Resource-Management
  - LoadAccelerator()
  - LoadBitmap()
  - LoadMenu()
- System
  - GetSystemMetrics()
- Text
  - SetTextColor()
  - TextOut()
- Class
  - RegisterClass()
- Windows
  - BeginPaint()
  - CreateWindow()
  - EndPaint()
  - GetWindowLong()

- InvalidateRect()
- SetWindowLong()
- ShowWindow()
- UpdateWindow()

4.4 Source Files

Table 3 lists all the source files used by the MDI sample program. You will find these files on the CD-ROM supplied with this book in the MDI directory and in its two subdirectories, OS2 and WIN32. The OS2 and WIN32 subdirectories contain respectively platform specific files for OS/2 and Windows while the MDI directory contains the files common to both OS/2 and Windows.

Table 3. Source Files

LOCATION	NAME	DESCRIPTION
MDI	MDI.H	Common source header file
MDI	MDI.C	Common source code file
MDI	OS2TILE.BMP	Common bitmap resource file
MDI OS2	MAIN.C	OS/2 specific source file
MDI OS2 and MDI WIN32	MDI.RC	OS/2 and Windows specific resource file
MDI OS2	MDI.MAK	OS/2 specific makefile
MDI WIN32	MAKEFILE	Windows specific makefile
MDI OS2 and MDI WIN32	MDI.DEF	OS/2 and Windows specific module definition file
MDI OS2 and MDI WIN32	MDI.ICO	OS/2 and Windows specific icon resource file

4.5 Coding

The coding takes place on Windows 95 or NT using Microsoft Visual C++ Version 4.0. This section gives some design and programming details of the MDI sample program. It is intended for application developers, in particular OS/2 application developers, who are not familiar with Microsoft MDI programming interface. Readers who have already developed applications using Microsoft MDI programming interface may wish to skip this section by going to 4.6, “Migration” on page 129.

In 4.5.1, “Resources” on page 111 we will look at the resources used by the application. In 4.5.2, “WinMain()” on page 112 to 4.5.4, “MDIWndProc()” on

page 118, we will talk about the main functions that make up the program with focus on what is unique to MDI programming.

4.5.1 Resources

The resources used by the MDI sample program are simple and are written in a straight forward manner using the Microsoft Developer's Studio. Figure 114 shows the resource definitions.

You will notice that there are two menus in the resource file. One, IDR_MENU, is simpler with only two pull-down menus, File and Help. It is used as the main MDI application menu. The other, IDR_MENU_CHILD, has two more pull-down menus, Draw and Window. It is used for the MDI child windows. How the MDI sample program switches between these two menus will be explained in 4.5.4, “MDIWndProc()” on page 118.

```
// Windows Resources
// Icon
IDI_ICON             ICON    DISCARDABLE    "mdi.ico"
IDI_DAPIE            ICON    DISCARDABLE    "dapie.ico"

// Bitmap
IDB_DAPIE            BITMAP  DISCARDABLE    "dapie.bmp"
// Main application menu
IDR_MENU MENU DISCARDABLE
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "&New",                   IDM_NEW
      MENUITEM "&Close",                  IDM_CLOSE, GRAYED
      MENUITEM SEPARATOR
      MENUITEM "&Exit",                  IDM_EXIT
   END
   POPUP "&Help"
   BEGIN
      MENUITEM "&About...",               IDM_ABOUT
   END
END
```

Figure 114 (Part 1 of 2). MDI Sample Program's Resources

```
// MDI child window menu
IDR_MENU_CHILD MENU DISCARDABLE
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "&New",                    IDM_NEW
        MENUITEM "&Close",                  IDM_CLOSE
        MENUITEM SEPARATOR
        MENUITEM "&Exit",                   IDM_EXIT
    END
    POPUP "&Draw"
    BEGIN
        MENUITEM "&Bitmap",                 IDM_BITMAP
        MENUITEM "&Graphics",               IDM_GRAPHICS
        MENUITEM "&Screen Capture",          IDM_SCREEN
        MENUITEM "&Text",                   IDM_TEXT
    END
    POPUP "&Window"
    BEGIN
        MENUITEM "&Cascade",                IDM_CASCADE
        MENUITEM "&Tile",                   IDM_TILE
        MENUITEM "Arrange &Icons",           IDM_ARRANGE
        MENUITEM "Close &All",              IDM_CLOSEALL
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About...",               IDM_ABOUT
    END
END

// About dialog
IDD_ABOUT DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_CAPTION | WS_SYSMENU
CAPTION "About DAPIE MDI Sample"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "OK",IDOK,67,74,50,14
    ICON          IDI_ICON,IDC_STATIC,163,72,18,20
    CTEXT         "DAPIE MDI Sample",IDC_STATIC,49,16,88,8
    ICON          IDI_DAPIE,IDC_STATIC,7,68,18,20
    CTEXT         "(C) Copyright IBM Corp. 1996",IDC_STATIC,45,36,96,8
    CTEXT         "Developed by IBM ITSC Austin",IDC_STATIC,41,56,103,8
END
```

Figure 114 (Part 2 of 2). MDI Sample Program's Resources

4.5.2 WinMain()

The WinMain() function is the entry point of Windows application programs. It usually does routine tasks such as global variables initialization, window classes registering, main application window creation and message dispatching. Figure 115 on page 113 lists the WinMain() section of the MDI sample program. In addition to the routine Windows application processing, the WinMain() function of the MDI sample program registers an extra window class for MDI child windows called MDIChild. The window

procedure of the MDIChild window class, MDIWndProc(), will be discussed in 4.5.4, “MDIWndProc()” on page 118.

The message loop processing in the MDI sample program differs a little from a normal message loop processing. This is because each MDI child windows has its own system menu with different accelerator keys from the main window. The WinMain() function must translate the accelerator keys by calling TranslateMDISysAccel() function. Notice the two different window handles, hwndFrame and hwndClient, respectively passed to TranslateAccelerator() and TranslateMDISysAccel(). hwndFrame is the MDI frame window handle and hwndClient the MDI Client window handle. TranslateAccelerator() translates the accelerator keys for hwndFrame while TranslateMDISysAccel() translates the accelerator keys for hwndClient.

```
//****************************************************************************
//* WinMain
//****************************************************************************
int APIENTRY WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
   WNDCLASS wc;
   HWND    hwndFrame, hwndClient;
   MSG     msg;        // message structure
   HANDLE   hAccelTable; // accelarator table handle.

   hInst = hInstance;

   // Register the main window class
   wc.style          = 0;
   wc.lpfnWndProc     = MainWndProc;
   wc.cbClsExtra      = 0;
   wc.cbWndExtra      = 0;
   wc.hInstance       = hInstance;
   wc.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON) );
   wc.hCursor         = LoadCursor(NULL, IDC_ARROW);
   wc.hbrBackground    = (HBRUSH)(COLOR_APPWORKSPACE);
   wc.lpszMenuName    = MAKEINTRESOURCE(IDR_MENU);
   wc.lpszClassName    = szAppName;
   if(!RegisterClass(&wc))
      return FALSE;
```

Figure 115 (Part 1 of 2). MDI Sample Program WinMain() Function

```
    // Register the MDI child window class
    wc.style          = 0;
    wc.lpfnWndProc     = MDIWndProc;
    wc.cbWndExtra      = 8;
    wc.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON) );
    wc.lpszMenuName    = NULL;
    wc.lpszClassName   = szMDIChild;
    if(!RegisterClass(&wc))
       return FALSE;

    // Create the main window - the MDI Frame window
    hwndFrame = CreateWindow( szAppName,
                      "DAPIE MDI Sample",
                      WS_OVERLAPPEDWINDOW | WS_CLIPCHILDREN,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      CW_USEDEFAULT, CW_USEDEFAULT,
                      NULL,
                      NULL,
                      hInstance,
                      NULL );

    // Get the MDI client window handle
    hwndClient = GetWindow( hwndFrame, GW_CHILD );

    // Show the main window
    ShowWindow( hwndFrame, nCmdShow );
    UpdateWindow( hwndFrame );

    // Load accelarator table.
    hAccelTable = LoadAccelerators( hInstance, MAKEINTRESOURCE(IDR_ACCEL) );

    // Dispatch the messages.
    while (GetMessage(&msg, NULL, 0, 0))
    {
       if( !TranslateAccelerator( hwndFrame, hAccelTable, &msg ) &&
          !TranslateMDISysAccel( hwndClient, &msg ))
       {
          TranslateMessage(&msg);
          DispatchMessage(&msg);
       }
    }

    return ( msg.wParam );
}
```

Figure 115 (Part 2 of 2). MDI Sample Program WinMain() Function

4.5.3 MainWndProc()

MainWndProc() is the window procedure of the MDI frame window. Its main responsibilities are to initialize the MDI client window and to process the menu commands. Figure 117 on page 116 lists the MainWndProc() section of the MDI sample program.

During the processing of the WM_CREATE message, MainWndProc() calls CreateWindow() to create the MDI client window. The window class name is set to MDClient, which is the preregistered class for MDI client windows. The last argument to CreateWindow() is set to a pointer to a structure of type CLIENTCREATESTRUCT. The structure contains the initial main menu handle and the first MDI child window's identifier.

MainWndProc() gets menu commands from the WM_COMMAND message and treats them according to their type. The menu commands concerning MDI windows are grouped into three types for processing as follows:

1. Actions to be applied on to a single MDI child window.
   - IDM_CLOSE: Close a MDI child window
   - IDM_BITMAP: Draw bitmaps in a MDI child window
   - IDM_GRAPHICS: Draw graphics in a MDI child window
   - IDM_SCREEN: Capture the screen image and draw it in a MDI child window
   - IDM_TEXT: Draw texts in a MDI child window

   MainWndProc() sends the WM_MDIGETACTIVE message to the MDI client window to get the currently active MDI child window's handle then forwards the commands to the active MDI child window.
2. Actions to be applied to all the MDI child windows.
   - IDM_CASCADE: Reorganize the MDI child windows on cascade
   - IDM_TILE: Reorganize the MDI child windows on tiles
   - IDM_ARRANGE: Reorganize the icons of the MDI child windows
   - IDM_CLOSEALL: Close all the MDI child windows

   These commands, except IDM_CLOSEALL, are translated into one of the following MDI messages, WM_MDITILE, WM_MDICASCADE or WM_MDIICONARRANGE, then sends the translated messages to the MDI client window for processing. For the IDM_CLOSEALL command, EnumChildWindows() function is called to pass the window handle of each MDI child window one after the other to the CloseMDIChild() function. This function then sends a WM_CLOSE message to each window handle it receives.
3. Actions to be applied to the MDI client window.

   It concerns the IDM_NEW command. On receipt of this command, MainWndProc() calls NewMDIChild() function to create a new MDI child window. Figure 116 on page 116 lists the NewMDIChild() section the MDI sample program.

```
// Create a new MDI child window
HWND NewMDIChild( HWND hwndClient, LONG idChild )
{
   CHAR szTitle[80];
   HWND hwndChild;
   MDICREATESTRUCT mdi;

   wsprintf( szTitle, "MDI Child %d", idChild );
   mdi.szClass = szMDIChild;
   mdi.szTitle = szTitle;
   mdi.hOwner  = NULL;
   mdi.x = mdi.y = mdi.cx = mdi.cy = CW_USEDEFAULT;
   mdi.style   = 0;
   mdi.lParam  = 0;
   hwndChild = (HWND)SendMessage( hwndClient, WM_MDICREATE, 0, (LPARAM)&mdi );

   return hwndChild;
}
```

Figure 116. NewMDIChild(): Create a New MDI Child Window

NewMDIChild() initializes a MDICREATESTRUCT structure and sends the MDI client window a WM_MDICREATE message with the address of this structure. The MDI client window then creates a new MDI child window.

Notice that all messages that MainWndProc() chooses not to process are passed to DefFrameProc() instead of DefWindowProc(). DefFrameProc() is a MDI specific function that forwards the unprocessed messages to the MDI client window.

```
//****************************************************************************
//* MainWndProc
//****************************************************************************
LRESULT CALLBACK MainWndProc( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static HWND hwndClient;
   static int iMDILast = 0;
```

Figure 117 (Part 1 of 3). MainWndProc(): Main Window Procedure

```
switch (message)
{
   case WM_CREATE:
   {
      HMENU hMenu = LoadMenu( hInst, MAKEINTRESOURCE(IDR_MENU) );
      CLIENTCREATESTRUCT client;
      client.hWindowMenu  = GetSubMenu( hMenu, 0 );
      client.idFirstChild = 1;
      hwndClient = CreateWindow( "MDIClient", NULL,
                        WS_CHILD | WS_CLIPCHILDREN | WS_VISIBLE,
                        0,0,0,0,
                        hwnd, NULL, hInst, (LPVOID)&client );
      break;
   }
   case WM_COMMAND:
      switch (LOWORD(wParam))
      {
         case IDM_NEW:
         {
            if ( iMDICount == 0) {
               HMENU hMenu = GetMenu( hwnd );
               EnableMenuItem( hMenu, IDM_CLOSE, MF_ENABLED );
               iMDILast = 0;
            } /* endif */
            iMDICount++;
            iMDILast++;
            NewMDIChild( hwndClient, iMDILast );
            break;
         }
         case IDM_CLOSE:
         case IDM_BITMAP:
         case IDM_GRAPHICS:
         case IDM_SCREEN:
         case IDM_TEXT:
         {
            HWND hwndChild = (HWND)SendMessage( hwndClient, WM_MDIGETACTIVE, 0, 0 );
            if( hwndChild )
               SendMessage( hwndChild, WM_COMMAND, wParam, lParam);
               break;
         }
         case IDM_EXIT:
            SendMessage( hwnd, WM_CLOSE, 0, 0 );
            break;

         case IDM_CASCADE:
            SendMessage( hwndClient, WM_MDICASCADE, 0, 0 );
            break;
```

Figure 117 (Part 2 of 3). MainWndProc(): Main Window Procedure

```
            case IDM_TILE:
               SendMessage( hwndClient, WM_MDITILE, 0, 0 );
               break;

            case IDM_ARRANGE:
               SendMessage(hwndClient, WM_MDIICONARRANGE, 0, 0 );
               break;

            case IDM_CLOSEALL:
               EnumChildWindows( hwndClient, CloseMDIChild, 0 );
               break;

            case IDM_ABOUT:
               DialogBox( hInst, MAKEINTRESOURCE(IDD_ABOUT), hwnd, AboutDlgProc );
               break;

            default:
               return DefFrameProc(hwnd, hwndClient, message, wParam, lParam);
      }
      break;

      case WM_DESTROY:
         PostQuitMessage(0);
         break;

      default:
         return DefFrameProc(hwnd,  hwndClient, message, wParam, lParam);
  }

  return 0;
}
```

Figure 117 (Part 3 of 3). MainWndProc(): Main Window Procedure

4.5.4 MDIWndProc()

MDIWndProc() is the MDI child window procedure. Its main goal is to draw the MDI child windows. Figure 119 on page 120 lists the MDIWndProc() section of the MDI sample program.

On receipt of the WM_CREATE message, MDIWndProc() calls InitMDIChild() function to initialize a new MDI child window. Figure 118 on page 119 lists the InitMDIChild() section of the MDI sample program.

```
// Initialize MDI Child Window
BOOL InitMDIChild( HWND hwnd )
{
   // Create a screen compatible memory DC
   HDC hdc = GetDC(hwnd);
   HDC hdcMem = CreateCompatibleDC( hdc );

   // Get the screen size
   int cx  = GetSystemMetrics(SM_CXSCREEN);
   int cy  = GetSystemMetrics(SM_CYSCREEN);

   // Create a screen size bitmap
   HBITMAP hBitmap = CreateCompatibleBitmap( hdc, cx, cy );

   // Get the gray brush
   HBRUSH hBrush = GetStockObject( LTGRAY_BRUSH );

   // Select the bitmap and the gray brush for the memory DC
   SelectObject( hdcMem, hBrush );
   SelectObject( hdcMem, hBitmap );

   // Fill the memory DC on gray
   PatBlt( hdcMem, 0, 0, cx, cy, PATCOPY );

   // Store the handles of the memory DC and the bitmap
   SetWindowLong( hwnd, 0, (LONG)hdcMem );
   SetWindowLong( hwnd, 4, (LONG)hBitmap );
   ReleaseDC( hwnd, hdc );

   return TRUE;
}
```

Figure 118. InitMDIChild(): Initilize a MDI Child Window

The MDI child window initialization consists of the following activities:

- Create a display compatible memory device context
- Create a screen size bitmap and store it into the memory device context
- Fill the memory device context with gray by calling the PatBlt() function
- Store the handles of the memory device context and the bitmap with the MDI child window for further use by the drawing functions and resource release functions when the window is closed

When the MDI child window becomes active upon receipt of the WM_MDIACTIVE message, MDIWndProc() replaces the main application menu with the MDI child menu.

Upon receipt of the close request (WM_CLOSE message), MDIWndProc() switches back to the main application menu and releases the memory device context and the bitmap created at initialization time.

On receipt of the drawing commands of IDM_BITMAP, IDM_GRAPHICS, IDM_SCREEN and IDM_TEXT, forwarded by the main window, MDIWndProc() calls one of the drawing functions to fill the memory device context of the MDI child window then invalidates its painting area.

The IDM_SCREEN command is treated in a special way: MDIWndProc() hides at first the main window in order to capture the entire screen image without the main window, then it starts a timer and waits. This allows for the redrawing the entire screen. Then on receipt of the WM_TIMER message, MDIWndProc() call CaptureScreen() to capture the screen image and fill the MDI child window's memory device context with it. All these drawing functions will be discussed in 4.5.5, “Drawing Functions” on page 122.

Each time a MDI child window is invalidated, a WM_PAINT message will be generated. On receipt of this message, MDIWndProc() gets the device context handle and the memory device context handle of the MDI child window, and copies the contents of the second into the first.

Similar to MainWndProc(), MDIWndProc() passes the unprocessed messages to DefMDIChildProc() instead of DefWindowdProc(). Like DefFrameProc(), DefMDIChildProc() is a MDI specific function.

```
//*************************************************************************
//* MDIWndProc
//*************************************************************************
LRESULT CALLBACK MDIWndProc( HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)
{
   static HWND hwndFrame, hwndClient;

   switch (message)
   {
      case WM_CREATE:
      {
         hwndClient = ((PCREATESTRUCT)lParam)->hwndParent;
         hwndFrame = GetParent( hwndClient );
         InitMDIChild( hwnd );
         break;
      }
```

Figure 119 (Part 1 of 3). MDIWndProc(): MDI Client Window Procedure

```
        case WM_MDIACTIVATE:
            if ( (HWND)lParam == hwnd )
            {
                HMENU hMenu = LoadMenu( hInst, MAKEINTRESOURCE(IDR_MENU_CHILD) );
                SendMessage( hwndClient, WM_MDISETMENU, (WPARAM)hMenu, (LPARAM)GetSubMenu(hMenu, 2) );
                DrawMenuBar( hwndFrame );
            }
            return 0;

        case WM_CLOSE:
        {
            HMENU hMenu = LoadMenu( hInst, MAKEINTRESOURCE(IDR_MENU) );
            HDC hdcMem = (HDC)GetWindowLong( hwnd, 0 );
            HBITMAP hBitmap = (HBITMAP)GetWindowLong( hwnd, 4 );
            SetMenu( hwndFrame, hMenu );
            DeleteDC( hdcMem );
            DeleteObject( hBitmap );
            iMDICount--;
            break;
        }
        case WM_COMMAND:
            switch( LOWORD( wParam ) )
            {
                case IDM_CLOSE:
                    SendMessage( hwnd, WM_CLOSE, 0, 0 );
                    break;

                case IDM_GRAPHICS:
                    DrawMyGraphics( hwnd );
                    break;

                case IDM_BITMAP:
                    DrawMyBitmap( hwnd );
                    break;

                case IDM_TEXT:
                    DrawMyText( hwnd );
                    break;

                case IDM_SCREEN:
                    ShowWindow( hwndFrame, SW_HIDE );
                    SetTimer( hwnd, 1, 1000, (TIMERPROC)MDIWndProc );
                    break;
            }
            break;

        case WM_TIMER:
            CaptureScreen( hwnd );
            ShowWindow( hwndFrame, SW_SHOWNORMAL );
            KillTimer( hwnd, 1 );
            break;
```

Figure 119 (Part 2 of 3). MDIWndProc(): MDI Client Window Procedure

```
      case WM_PAINT:
      {
         PAINTSTRUCT  ps;
         HDC hdc, hdcMem;
         hdc = BeginPaint( hwnd, &ps );
         hdcMem = (HDC)GetWindowLong( hwnd, 0 );
         BitBlt( hdc, ps.rcPaint.left, ps.rcPaint.top,
                ps.rcPaint.right - ps.rcPaint.left,
                ps.rcPaint.bottom - ps.rcPaint.top,
                hdcMem, ps.rcPaint.left, ps.rcPaint.top, SRCCOPY );
         EndPaint( hwnd, &ps );
         break;
      }
      default:
         return DefMDIChildProc(hwnd, message, wParam, lParam);
   } //switch

   return DefMDIChildProc(hwnd, message, wParam, lParam);
}
```

Figure 119 (Part 3 of 3). MDIWndProc(): MDI Client Window Procedure

4.5.5 Drawing Functions

The basic design of the drawing functions of the MDI sample program is to draw in the memory device context associated with a MDI child window then invalidate the MDI child window to generate the WM_PAINT message. On receipt of this message, MDIWndProc() copies the contents of the memory device context into the the display device context of the MDI child window. Each drawing function begins with a request of the memory device context handle of the MDI child window and ends with invalidating the MDI child window.

4.5.5.1 Draw Bitmap

Figure 120 on page 123 gives the listing of the bitmap drawing function, DrawMyBitmap() and Figure 121 on page 124 shows the drawing results in a MDI child window.

DrawMyBitmap() creates a temporary memory device context, hdcBitmap, loads a bitmap into it, then fills the MDI child window's memory device context, hdcMem, with the contents.

```
// Draw a bitmap in the MDI child window's memory DC
BOOL DrawMyBitmap( HWND hwnd )
{
   HDC hdcMem, hdcBitmap;
   HBITMAP hBitmap;
   BITMAP bmp;
   int x, y, cx, cy;

   // Get the MDI child window memory DC
   hdcMem = (HDC)GetWindowLong( hwnd, 0 );

   // Create a bitmap DC and load a bitmap in
   hdcBitmap = CreateCompatibleDC( hdcMem );
   hBitmap = LoadBitmap( hInst, MAKEINTRESOURCE(IDB_DAPIE) );
   GetObject( hBitmap, sizeof(BITMAP), (LPSTR) &bmp ) ;
   SelectObject( hdcBitmap, hBitmap );
   cx = GetSystemMetrics(SM_CXSCREEN);
   cy = GetSystemMetrics(SM_CYSCREEN);

   // Copy the bitmap DC in the MDI child window memory DC
   for( x=0; x < cx; x+= bmp.bmWidth )
      for( y = 0; y < cy; y+= bmp.bmHeight )
         BitBlt( hdcMem, x, y, bmp.bmWidth, bmp.bmHeight, hdcBitmap, 0, 0,SRCCOPY);

   // Release resource and invalidate the MDI child window
   DeleteDC( hdcBitmap );
   DeleteObject( hBitmap );
   InvalidateRect( hwnd, NULL, TRUE );

   return TRUE;
}
```

Figure 120. DrawMyBitmap(): Draw Bitmaps

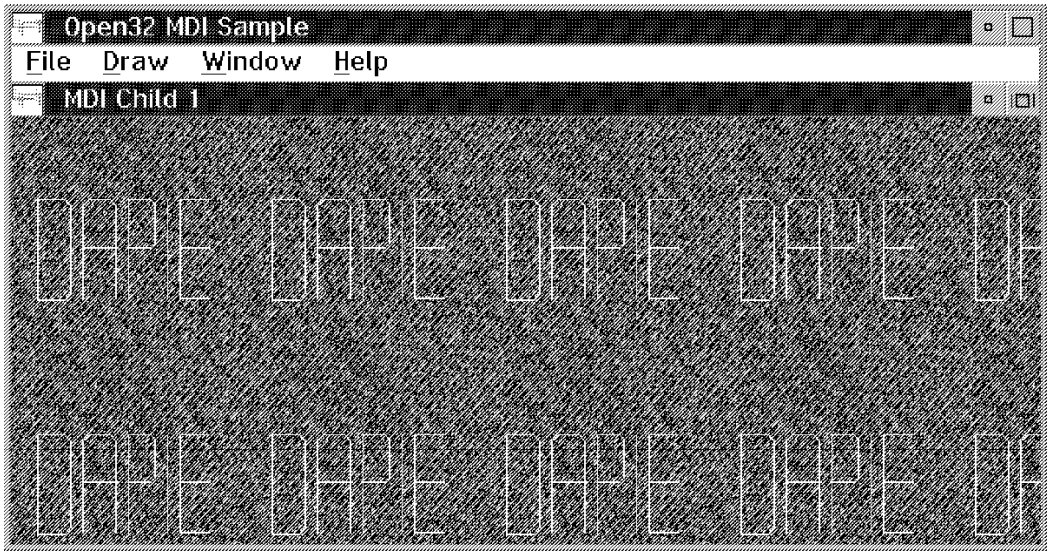


Figure 121. Bitmaps

4.5.5.2 Draw Graphics

Figure 122 gives the listing of the graphics drawing function, DrawMyGraphics() and Figure 123 on page 125 shows the drawing results.

```
// Draw graphics
BOOL DrawMyGraphics( HWND hwnd )
{
   HDC hdcMem = (HDC)GetWindowLong( hwnd, 0 );
   int i, r, g, b, x, y, cx, cy;
   HBRUSH hBrush, hOldBr;

   // Fill the MDI child memory DC on gray
   PaintDCGray( hdcMem );
```

Figure 122 (Part 1 of 2). DrawMyGraphics(): Draw Graphics

```
    // Draw 10 ellipses with random coordinates, size and color
    for( i = 0; i < 10; i++ )
    {
        r = rand()%256; g = rand()%256; b = rand()%256;
        x = rand()/100; y = rand()/100;
        cx = rand()/100; cy = rand()/100;
        hBrush = CreateSolidBrush(RGB(r,g,b));
        hOldBr = SelectObject( hdcMem, hBrush );
        Ellipse( hdcMem, x, y, x+cx, y+cy );
        SelectObject( hdcMem, hOldBr );
        DeleteObject( hBrush );
    }
    InvalidateRect( hwnd, NULL, TRUE );

    return TRUE;
}
```

Figure 122 (Part 2 of 2). DrawMyGraphics(): Draw Graphics

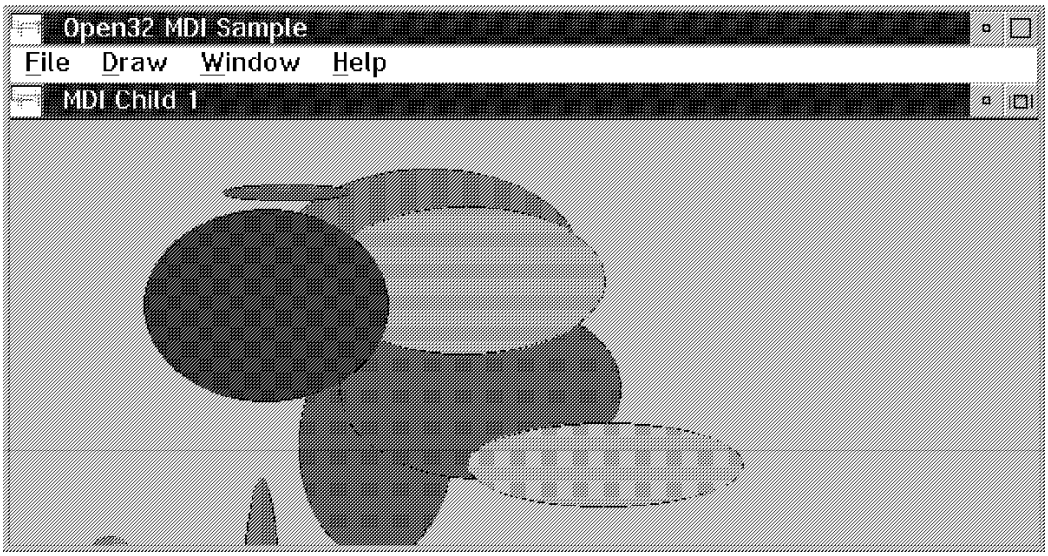


Figure 123. Graphics

DrawMyGraphics() fills the MDI child window's memory device context, hdcMem, with gray color. Then it draws 10 ellipses of random size, position and color.

4.5.5.3 Draw Text

Figure 124 on page 126 gives the listing of the text drawing function, DrawMyText().

```
// Draw Text
BOOL DrawMyText( HWND hwnd )
{
   LOGFONT   lf;
   CHOOSEFONT cf;
   HFONT     hFont;
   HFONT     hFontOld;
   int y, r, g, b;
   HDC hdc, hdcMem;
   hdc = GetDC( hwnd );

   // Fill in CHOOSEFONT structure
   memset(&cf, 0, sizeof(CHOOSEFONT));
   cf.lStructSize = sizeof(CHOOSEFONT);
   cf.hwndOwner = hwnd;
   cf.hDC = hdc;
   cf.lpLogFont = &lf;
   cf.iPointSize = 120;
   cf.Flags = CF_SCREENFONTS | CF_EFFECTS | CF_LIMITSIZE;
   cf.rgbColors = RGB(0, 0, 0);
   cf.lCustData = 0L;
   cf.lpfnHook = NULL;
   cf.lpTemplateName = (LPSTR)NULL;
   cf.hInstance = NULL;
   cf.lpszStyle = (LPSTR)NULL;
   cf.nFontType = SCREEN_FONTTYPE;
   cf.nSizeMin = 6;
   cf.nSizeMax = 48;

   // Show the font selection diolog
   if( ChooseFont(&cf) )
   {
      // Create a font base on the selection
      hFont = CreateFontIndirect(cf.lpLogFont);

      // Get the MDI child window's memory DC and fill it on gray
      hdcMem = (HDC)GetWindowLong( hwnd, 0 );
      PaintDCGray( hdcMem );

      // Select the font and set drawing modes
      hFontOld = SelectObject( hdcMem, hFont );
      SetMapMode( hdcMem, MM_TEXT );
      SetBkMode( hdcMem, TRANSPARENT );
```

Figure 124 (Part 1 of 2). DrawMyText(): Draw Text

```
      // Draw 8 strings with the same texts but different color
      for ( r = 0, y = 0; r < 0x100; r += 0xff) {
         for ( g = 0; g < 0x100; g += 0xff ) {
            for ( b = 0; b < 0x100; b += 0xff ) {
               SetTextColor( hdcMem, RGB(r, g, b));
               TextOut( hdcMem, 0, y, "Welcome to the Open32 World!", 28 );
               y += lf.lfHeight + 5;
            } /* endfor */
         } /* endfor */
      } /* endfor */
      SelectObject( hdcMem, hFontOld );
      DeleteObject( hFont );
      InvalidateRect( hwnd, NULL, TRUE );
   }
   ReleaseDC( hwnd, hdc );

   return TRUE;
}
```

Figure 124 (Part 2 of 2). DrawMyText(): Draw Text

DrawMyText() first prompts you to choose a font with the ChooseFont() common dialog box, as shown in Figure 125.

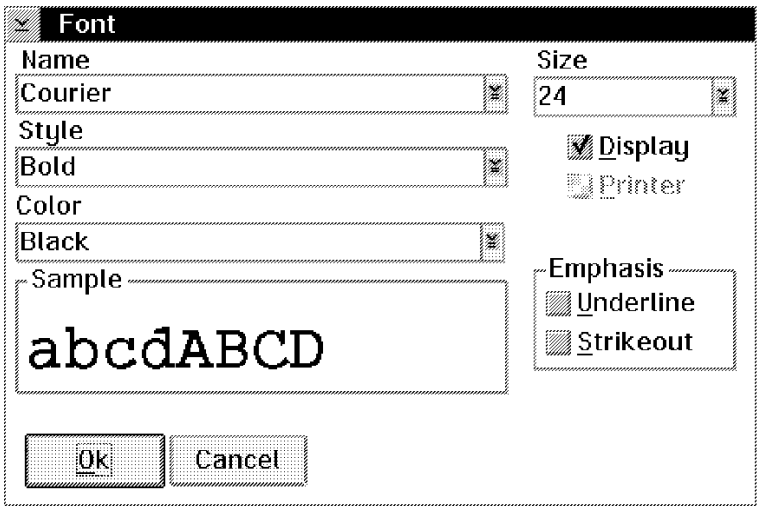


Figure 125. Font Selection Dialog

Once a font is chosen, DrawMyText() draws the same text string eight times with each text string in a different color. Figure 126 on page 128 shows the drawing results.

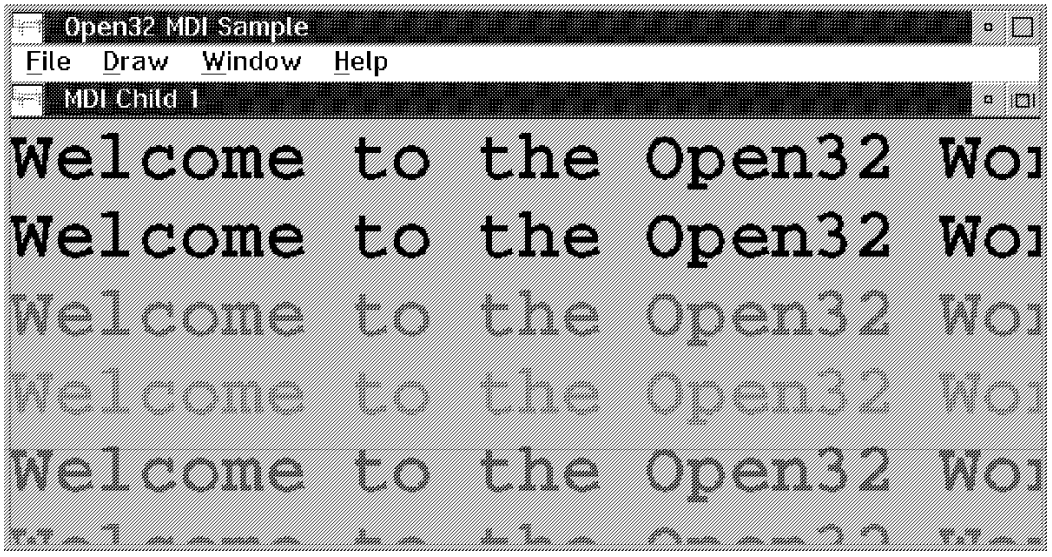


Figure 126. MDI Text Child Window

4.5.5.4 Capture Screen Image

Figure 127 gives the listing of the screen capture function, CaptureScreen() and Figure 128 on page 129 shows the drawing results.

```
// Capture screen image
BOOL CaptureScreen( HWND hwnd )
{
   // Get the memory DC
   HDC hdcMem = (HDC)GetWindowLong( hwnd, 0 );

   // Create a screen DC
   HDC hdcScreen = CreateDC( "DISPLAY", NULL, NULL, NULL );

   // Get the screen size
   int cx = GetSystemMetrics(SM_CXSCREEN);
   int cy = GetSystemMetrics(SM_CYSCREEN);
```

Figure 127 (Part 1 of 2). CaptureScreen(): Capture Screen Image

```
    // Copy the screen DC in the memory DC
    BitBlt( hdcMem, 0, 0, cx, cy, hdcScreen, 0, 0,SRCCOPY);

    // Delete the screen DC
    DeleteDC( hdcScreen );

    // Invalidate the MDI child window
    InvalidateRect( hwnd, NULL, TRUE );

    return TRUE;
}
```

Figure 127 (Part 2 of 2). CaptureScreen(): Capture Screen Image

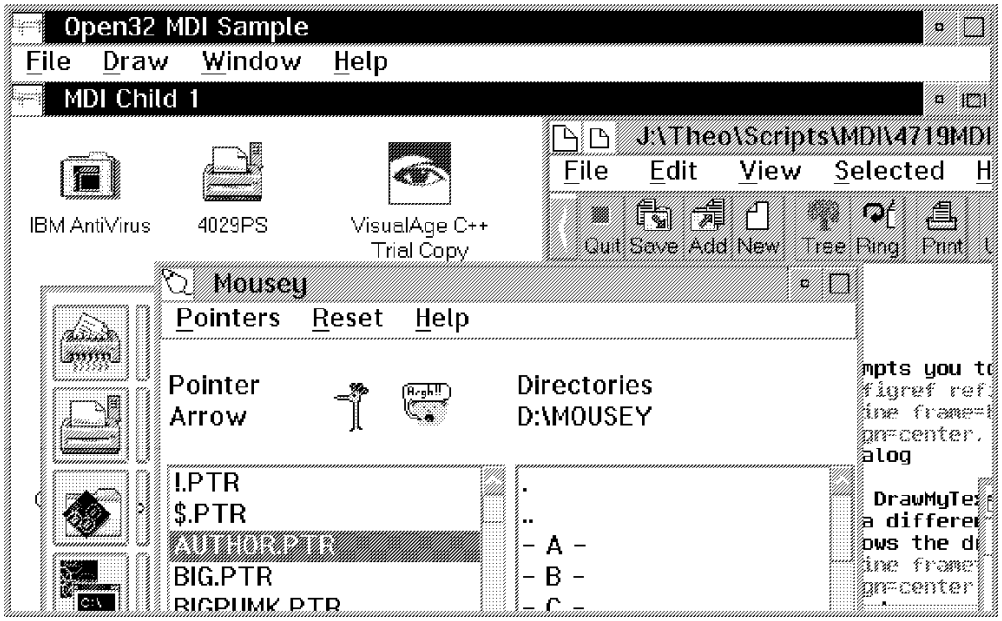


Figure 128. MDI Screen Capture Child Window

As explained in 4.5.4, “MDIWndProc()” on page 118, CaptureScreen() is called on receipt of the WM_TIMER message. It simply opens a Display device context and copies the contents of the screen image into the MDI child window's memory device context.

4.6 Migration

After coding and testing the MDI sample program in the Windows environment, we can port the program to OS/2. Since the MDI sample program is a common source code application, the migrating steps are

exactly the same as the ones described in Chapter 3, “Howdy, World!” on page 79:

- Convert the bitmap file dapie.bmp
- Convert the icon file dapie.ico
- Convert the resource file mdi.rc
- Make the system header files changes in the mdi.c file

  from:
  ```
  #include <windows.h>
  #include <commdlg.h>
  ```

  to: // system header files
  ```
  #ifdef OS2              // for OS2
  #include <os2win.h>
  #else                   // for Win32
  #include <windows.h>
  #include <commdlg.h>
  #endif
  ```

  The value OS2 is a precompiler variable defined for the VisualAge C++ compiler in the OS/2 MAKEFILE.
- Copy the main.c file from OS/2 Warp Toolkit
- Create a new module definition file
- Create a project makefile
- Compile the resources
- Compile the C source files
- Link the application and bind the resources

Figure 129 shows the DEF file for the MDI sample program to be used for OS/2.

Figure 130 on page 131 shows the MAKEFILE for the MDI sample program to be used for OS/2.

```
// mdi.def :DEF file for OS/2
NAME       MDI WINDOWAPI
DESCRIPTION 'Open32 MDI Sample Prgram (c) IBM, 1996'
STACKSIZE   65536
```

Figure 129. MDI Sample Program's DEF File

```
// makefile : MAKEFILE for OS/2
proj = mdi
cflags = /C /DOS2 /N3 /Ss /Ti /Wgen /Wpro
lflags = /CO
llib   = pmwinx.lib

$(proj).exe : $*.obj main.obj $*.res
  ilink $(lflags) $*.obj main.obj $(llib) $*.def
  rc $*.res $*.exe

$(proj).obj: $*.c resource.h
  icc $(cflags) $*.c

main.obj: $*.c
  icc $(cflags) $*.c

$(proj).res: $*.rc resource.h
  rc -r $*.rc
```

Figure 130. MDI Sample Program's MAKEFILE

Note

The debug compile and linking options are used. Once the program is tested, you can remove them.

Figure 131 on page 132 shows the MDI sample program file structure and the operating system dependent headers libraries and resources for building both the OS/2 and Win32 executable programs from the same common source code.

The MDI and HOWDY sample programs have demonstrated the techniques involved in developing Open32 applications which share a common code base between the OS/2 and Windows programs. In the upcoming chapters we will discuss how to work with programs which do not completely shared the same source code.

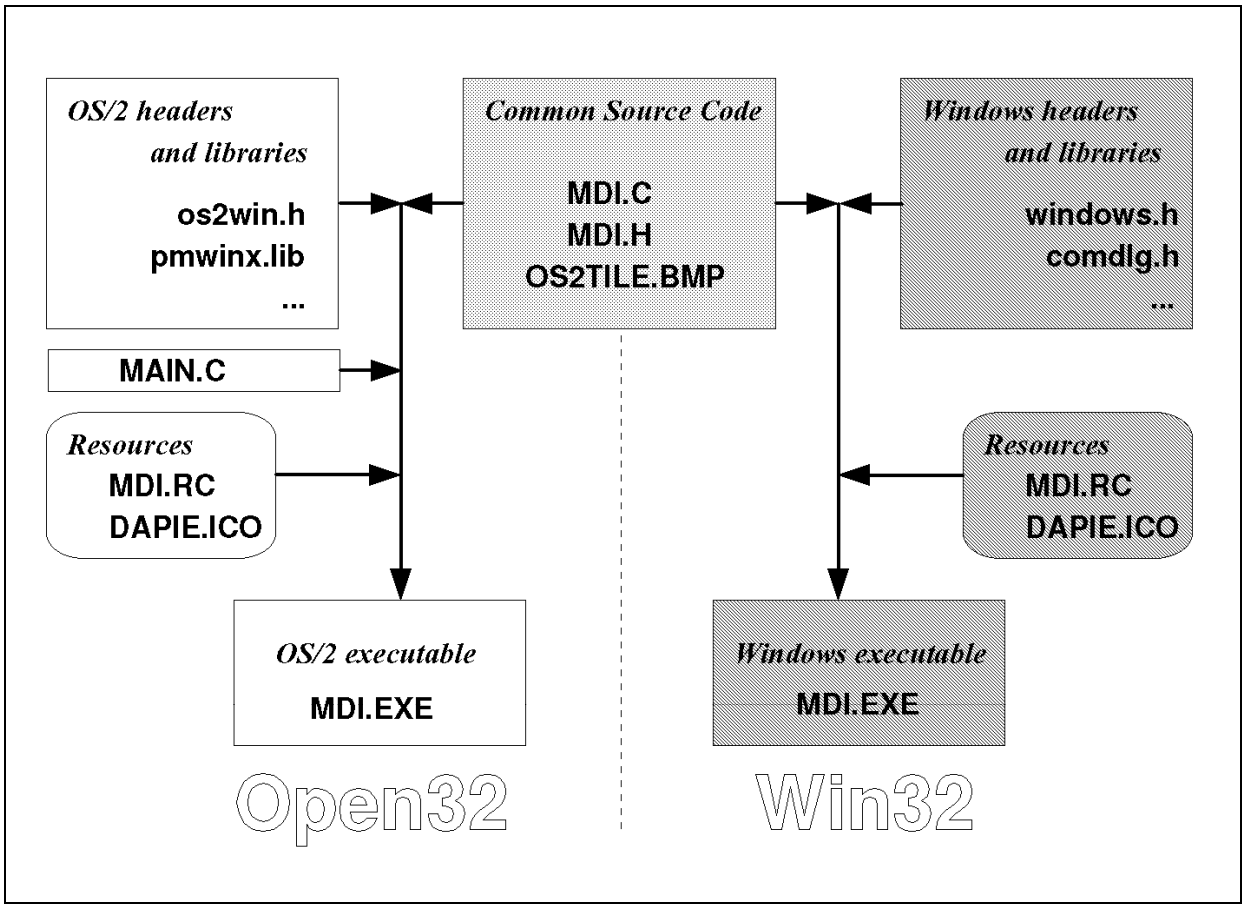


Figure 131. MDI Sample Program's File Structure

Chapter 5. Mixed Mode Sample Program

This chapter describes the mixed mode sample program. This sample program is a mixed mode application which uses a Win32 window class unsupported by Open32 in the Windows version of the program and uses an equivalent OS/2 window class in the OS/2 version of the program.

The mixed mode sample program was a new application development effort and this chapter will discuss the programming techniques used to design and code the application such that it has a common code base along with platform dependent code. The platform dependent part is combined with the common code base to build the application program for either the OS/2 or Win32 environment.

5.1 Application Overview

The sample program described in this chapter looks quite simple. It opens a main application window and displays a message in the center of the main window, as shown in Figure 132.

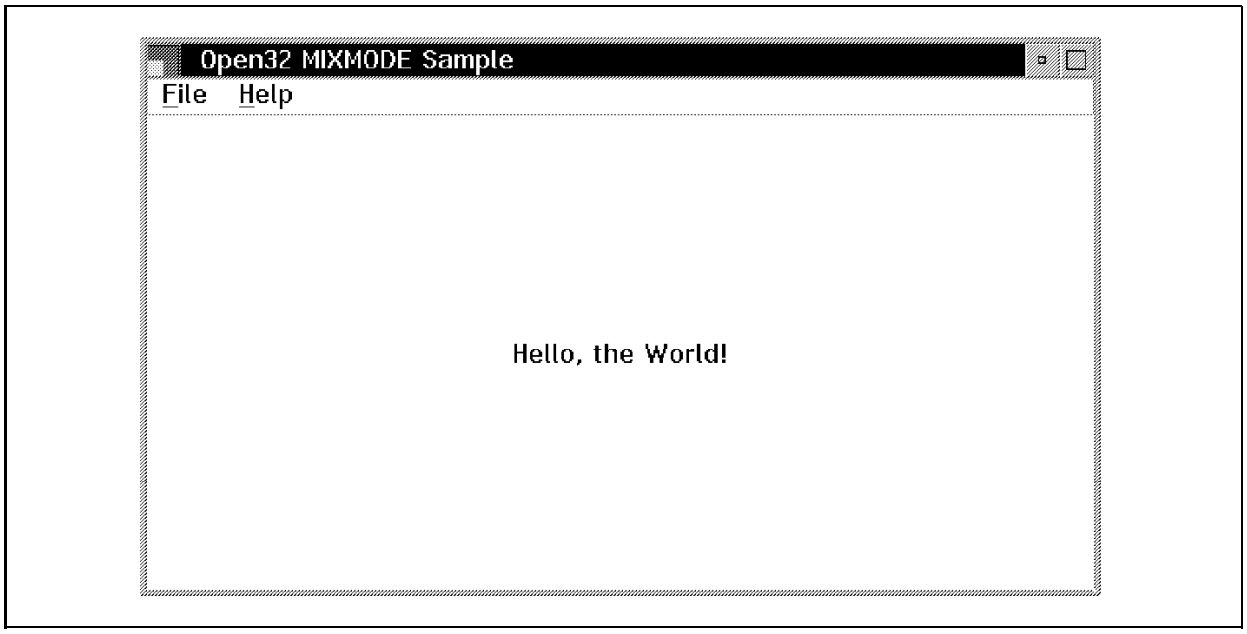


Figure 132. Mixed Mode Sample Program Overview

You can change the message's properties from the File pull-down menu using the Properties option, as shown in Figure 133 on page 134.

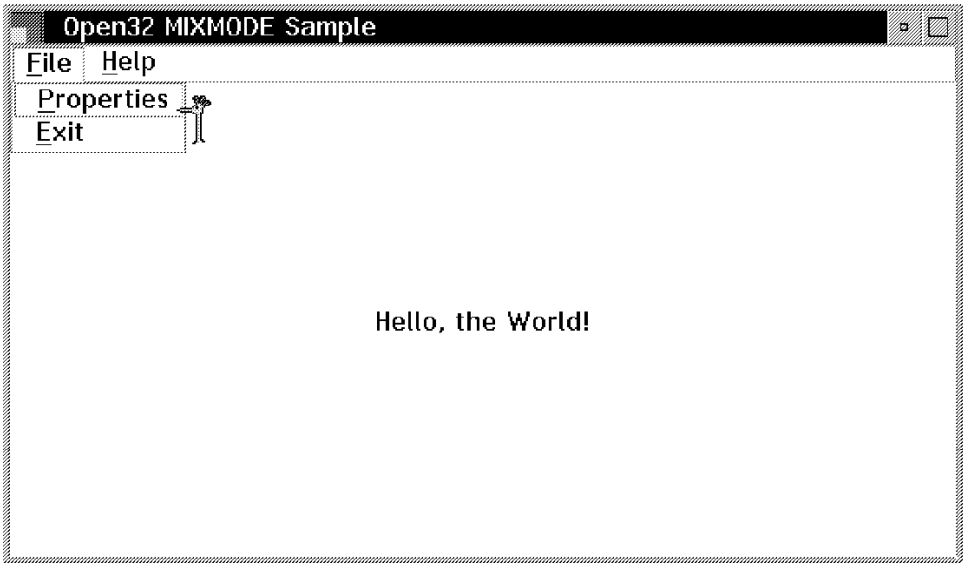


Figure 133. Mixed Mode Sample Program Main Menu

The message's properties are its text and color. By selecting the Properties option of the File pull-down, you can open a dialog box. The dialog box contains a Tab control in the Win32 version. Since the Tab control is not supported by Open32, when the application is migrated to OS/2, it needs to be replaced. In the mixed mode sample program, we have chosen to use an OS/2 notebook control to replace the Tab control.

These two controls provide the end user with the same functions but the implementation is very different as you will see in the mixed mode sample program.

Note

For simplicity, we will use the term *Notebook* interchangeably for Win32 Tab control and OS/2 notebook control.

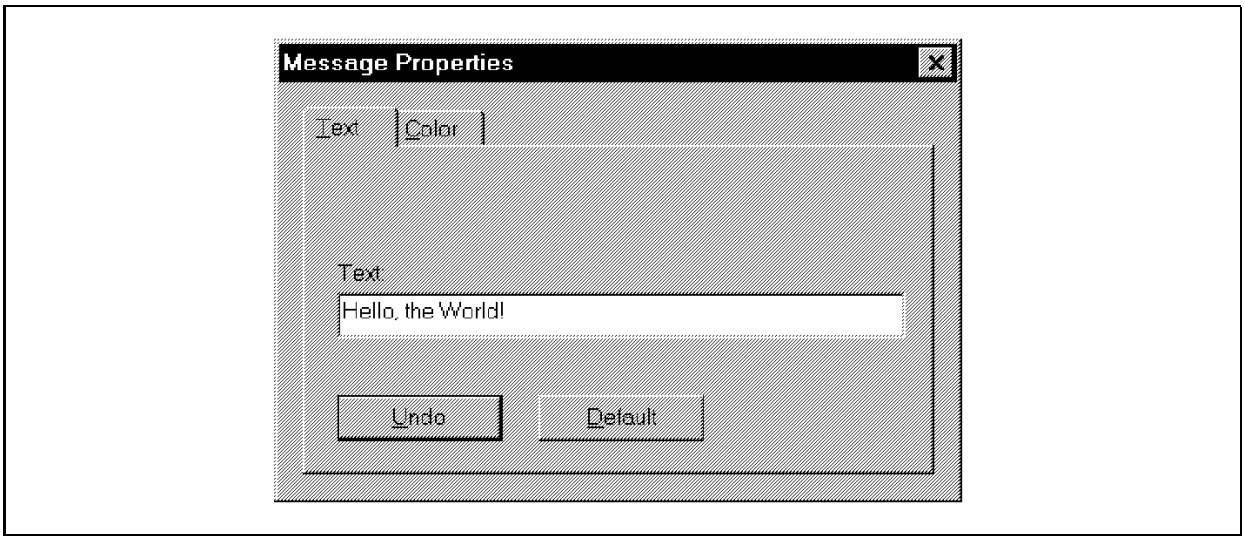


Figure 134. Mixed Mode Message Properties Tab Control on Windows 95

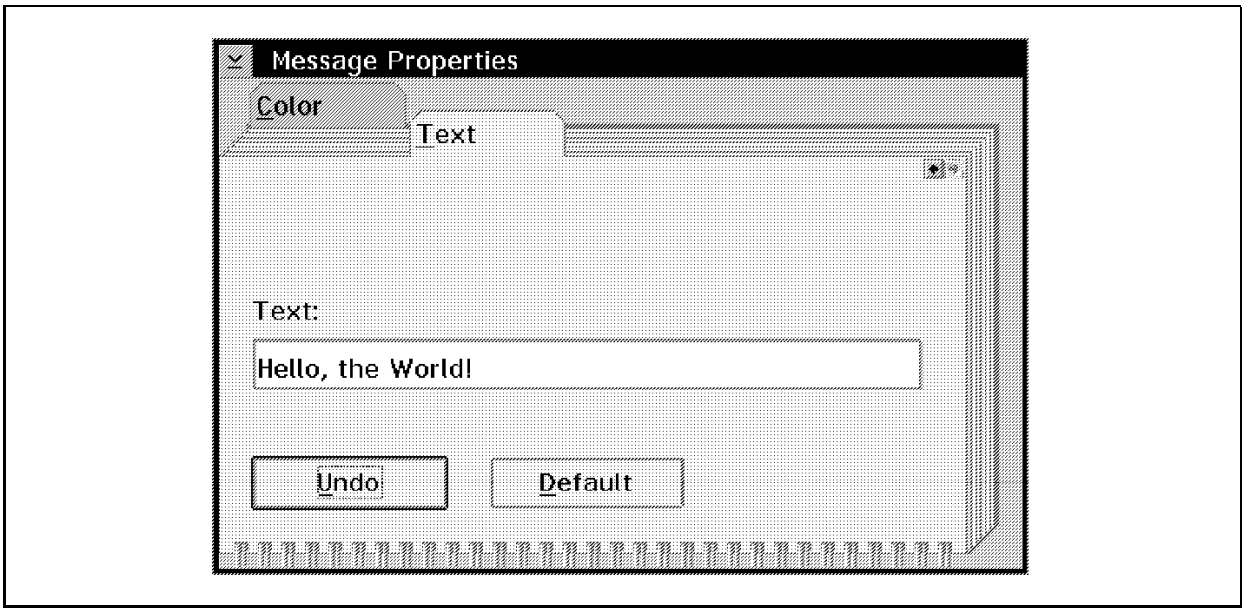


Figure 135. Mixed Mode Message Properties Notebook on OS/2

Figure 134 shows the text properties of the mixed mode sample program on the Windows 95 platform using the Win32 Tab control. Figure 135 for comparison shows the text properties of the sample using the Notebook control on the OS/2 Warp platform. Figure 136 on page 136 and Figure 137 on page 136 show the color properties on the Windows 95 and OS/2 platforms respectively. You can select the text and color tabs and modify the text and color of the message. The changes will become effective when

you close the Message Properties Dialog. You can use the Undo pushbutton to cancel the changes or the Default pushbutton to set the default text and color at any time. This will override any changes you have made using the message properties dialog.

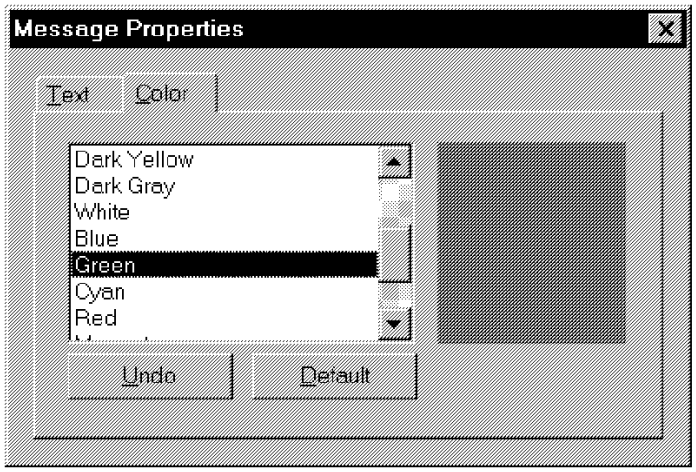


Figure 136. Mixed Mode Color Page on Windows 95

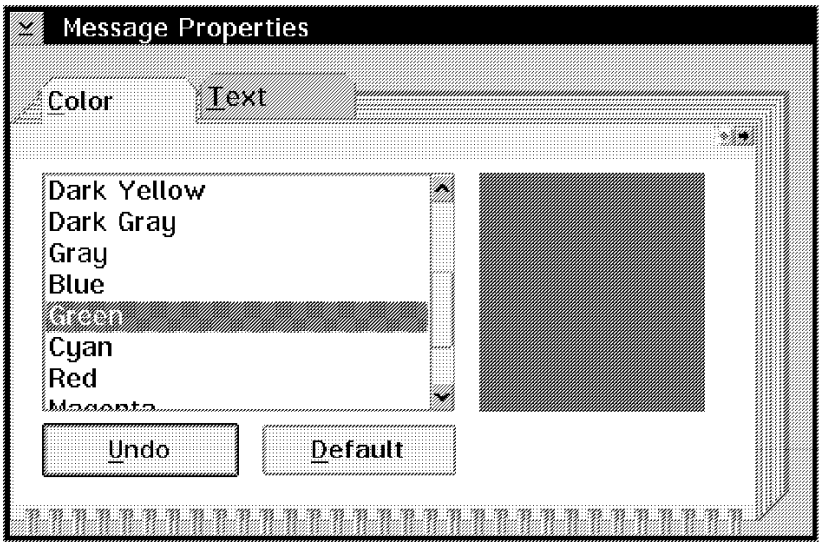


Figure 137. Mixed Mode Color Page on OS/2

5.2 Source Files

Table 4 lists all the source files used by the mixed mode sample program. You will find these files on the CD-ROM in this redbook in the MIXMODE directory and in its two subdirectories, OS2 and WIN32. The OS2 and WIN32 subdirectories contain platform specific files for OS/2 and Windows. The MIXMODE directory contains the files common to both OS/2 and Windows.

Table 4. Mixed Mode Sample Program Source Files

LOCATION	NAME	DESCRIPTION
MIXMODE	MIXMODE.C	Common source code file
MIXMODE	RESOURCE.H	Common resource header file
MIXMODE OS2	MAIN.C	OS/2 specific source file
MIXMODE OS2 and MIXMODE WIN32	DEPEND.C	OS/2 and Windows specific source code file
MIXMODE OS2 and MIXMODE WIN32	DEPEND.H	OS/2 and Windows specific header file
MIXMODE OS2 and MIXMODE WIN32	MIXMODE.RC	OS/2 and Windows specific resource file
MIXMODE OS2	MAKEFILE	OS/2 specific makefile
MIXMODE WIN32	MIXMODE.MAK	Windows specific makefile
MIXMODE OS2	MIXMODE.DEF	OS/2 specific module definition file
MIXMODE OS2 and MIXMODE WIN32	MIXMODE.ICO	OS/2 and Windows specific icon resource file

5.3 Application Design

The mixed mode sample program has five components. They are:

- Main application window
- Message properties dialog box
- Notebook control
- Text dialog box
- Color dialog box

as shown in Figure 138 on page 138.

Since the notebook control is platform specific, the separation between the common and platform specific source code is between the application main window and the message property dialog box. As a result, the application

main window code goes into the common source code base, MIXMODE.C file, along with the application entry point function WinMain(), while the other components are implemented in a platform specific file, DEPEND.C. Two functions, GetGMessage() and SetGMessage(), are defined to interface between the common and platform specific source codes. They isolate the platform specific features from the common source code base. These two functions are discussed in 5.4, “Coding.”

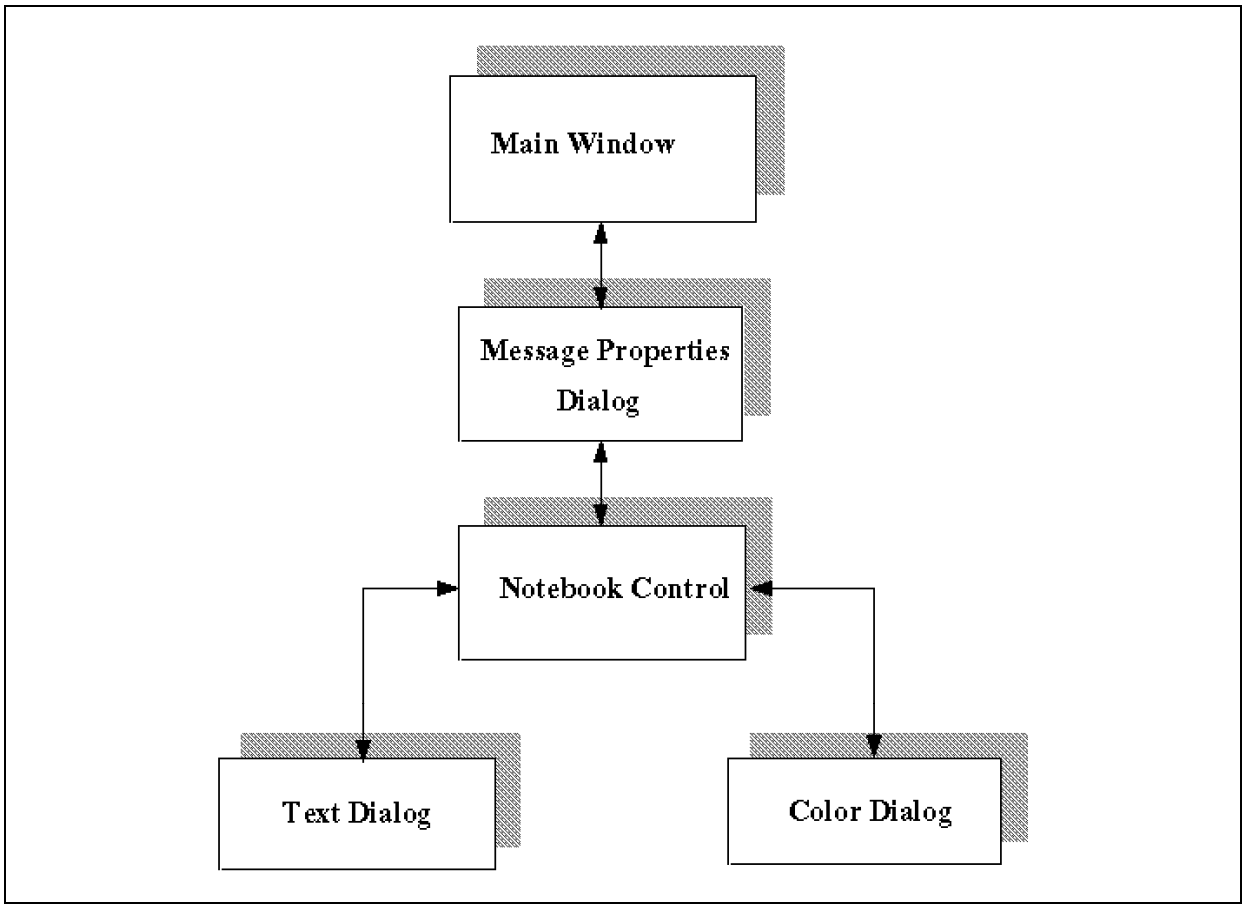


Figure 138. Mixed Mode Sample Program Architecture

5.4 Coding

The coding can be done on Windows 95 or NT. Microsoft Visual C++ Version 4.0 development tools can be used to develop the program. In this section we will discuss the coding that was done uniquely for the mixed mode sample program.

5.4.1 Resources

The resource file for the Windows version of the mixed mode application can be found in the MIXMODE.RC file in the WIN32 subdirectory of the CD-ROM in this redbook. The resource file defines the resources used by the application including the message property dialog box with the tab control, the text and color dialog boxes. The window procedures for these dialog boxes will be discussed in 5.4.3, “Platform Specific Code” on page 141. For comparison the OS/2 version of this resource file will be covered later in this chapter in 5.5.3, “Converting Platform Specific Source Code” on page 151. A copy of the Windows version is shown in Figure 139.

```
// mixmode.rc : Mixed Mode Sample Program's Resources
// Icon
IDI_ICON             ICON    DISCARDABLE     "mixmode.ico"
IDI_DAPIE            ICON    DISCARDABLE     "dapie.ico"

// Menu
IDR_MENU MENU DISCARDABLE
BEGIN
   POPUP "&File"
   BEGIN
      MENUITEM "&Properties",              IDM_PROP
      MENUITEM "&Exit",                   IDM_EXIT
   END
   POPUP "&Help"
   BEGIN
      MENUITEM "&About...",                IDM_ABOUT
   END
END

// About Dialog
IDD_ABOUT DIALOG DISCARDABLE  0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About Mixed Mode Sample"
FONT 8, "MS Sans Serif"
BEGIN
   DEFPUSHBUTTON   "OK",IDOK,67,74,50,14
   ICON          IDI_ICON,IDI_ICON,160,68,18,20
   LTEXT         "Open32 MIXMODE Sample",IDC_STATIC,54,18,78,8
   ICON          IDI_DAPIE,IDC_STATIC,7,68,18,20
   CTEXT         "(C) Copyright IBM Corp. 1996",IDC_STATIC,47,37,92,8
   CTEXT         "Developed by IBM ITSC Austin",IDC_STATIC,41,56,103,8
END
```

Figure 139 (Part 1 of 2). Mixed Mode Sample Program's Resources (MIXMODE.RC)

```
// Message Property dialog
IDD_BOOK DIALOG DISCARDABLE  0, 0, 205, 125
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL        "Tab1",IDC_NOTEBOOK,"SysTabControl32",0x0,7,7,191,111
END

// Text dialog
IDD_TEXT DIALOG DISCARDABLE  2, 14, 186, 94
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON   "&Undo",IDC_UNDO,7,73,50,14
    PUSHBUTTON     "&Default",IDC_DEFAULT,68,73,50,14
    LTEXT          "Text:",IDC_STATIC_TEXT,7,32,17,8
    EDITTEXT       IDC_EDIT_TEXT,7,42,172,14,ES_AUTOHSCROLL
END

// Color dialog
IDD_COLOR DIALOG DISCARDABLE  2, 14, 183, 92
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    PUSHBUTTON     "&Undo",IDC_UNDO,7,71,50,14
    PUSHBUTTON     "&Default",IDC_DEFAULT,63,71,50,14
    LISTBOX        IDC_LIST_COLOR,7,7,105,61,LBS_NOINTEGRALHEIGHT |
                  WS_VSCROLL | WS_TABSTOP
    LTEXT          "",IDC_STATIC_COLOR,119,7,57,61
END
```

Figure 139 (Part 2 of 2). Mixed Mode Sample Program's Resources (MIXMODE.RC)

5.4.2 Common Source Code

The common source code is in the file MIXMODE.C which can be found in the MIXMODE directory of the CD-ROM in this redbook. It is similar to the Open32 applications described in Chapter 3, “Howdy, World!” on page 79 and Chapter 4, “MDI Sample Program” on page 103. You may wish to reference the MIXMODE.C common source code to see how the application design that follows was actually coded in the program.

The program contains three functions. They are:

- Application entry point WinMain()
- Main window procedure MainWndProc()
- About dialog procedure AboutDlgProc()

Instead of processing the Properties dialog function in this code, the program uses the interface functions, GetGMessage() and SetGMessage() to communicate with the dependent code section where these functions are

performed. The prototypes of these functions can be seen in the header file DEPEND.H shown in Figure 140 on page 141.

MainWndProc() stores the message properties in a private static structure variable, named gmessage, of type GMESSAGE and uses it to exchange data with the GetGMessage() and SetGMessage() functions. GMESSAGE data structure has two fields: one field to hold the message text and one field to hold the message color.

On receipt of the MDI WM_CREATE message, the main window procedure, MainWndProc(), calls the interface function, GetGMessage(), to get the default values of the message properties.

On receipt of the Properties option of the File pull-down menu the WM_COMMAND message, MainWndProc() calls the SetGmessage() to open the message properties dialog box. It then waits for the return of the new message properties values.

On receipt of the WM_PAINT message, MainWndProc() retrieves the message property values in the variable gmessage. It then draws the message in the client area with the text and color found in the variable gmessage.

```
// mixmode.h

// Greeting message properties data type
typedef struct _GMESSAGE {
   CHAR  szText[64];      // Message text
   ULONG ulColor;         // Message text color
} GMESSAGE, *PGMESSAGE;

// Greeting message function prototypes
BOOL GetGMessage( PGMESSAGE pgmsg );
BOOL SetGMessage( HWND hwnd, PGMESSAGE pgmsg );
```

Figure 140. Mixed Mode Interface Data Type and Function Prototypes (DEPEND.H)

5.4.3 Platform Specific Code

The platform specific code is contained in the C code file DEPEND.C. The Windows version of this dependent code can be found in the WIN32 subdirectory of the MIXMODE directory on the CD-ROM in this redbook. For reference in the discussion of this code that follows a copy of the Windows version of DEPEND.C is shown in Figure 141 on page 144.

- GetGMessage() and SetGMessage()

These two functions provide the interface between the common and platform specific code. The GetGmessage() function simply fills in the input structure with the default value for the message properties and returns. The SetGMessage() records the input structure address in a static global variable to make it visible to all the private functions in the file. It then loads the message property dialog box to get user's input.

- BookDlgProc()

  BookDlgProc() is the window procedure of the message property dialog box. Its main role is to insert the text and the color pages into its notebook. It is done by the function InitBook().

  A page in a notebook is in fact a dialog box. The text and the color pages are respectively the text and the color dialog boxes.

  When the message property dialog box is closed by the user, the notebook will be destroyed and a WM_DESTROY message will be sent to the text and color dialog boxes.

- TextDlgProc()

  TextDlgProc() is the window procedure of the text dialog box. The text dialog box has an edit field. On receipt of the WM_INITDIALOG message, TextDlgProc() initializes the edit field with the last edited message text stored in the input GMESSAGE data structure. Remember the text field in this data structure was filled with the default message text when the GetGMessage() was called.

  On receipt of the pushbutton command WM_COMMAND->UNDO, TextDlgProc() restores the edit field with its initial text.

  On receipt of the pushbutton command WM_COMMAND->DEFAULT, TextDlgProc() restores the edit field with the default text.

  On receipt of the WM_DESTROY message, TextDlgProc() retrieves the text in the edit field and puts it in the text field of the input GMESSAGE data structure.

- ColorDlgProc()

  ColorDlgProc() is the window procedure of the color dialog box. The color dialog box uses a list box to present the 16 color choices and a static control to show the color currently selected. ColorDlgProc() is based on a few static global variables:

  - aszColor[]

    aszColor[] is an array of 16 text strings. Each element in the array contains the name of a color from black to white.

  - aulColor[]

aulColor[] is an array of 16 ULONG (unsigned long integer). Each element in the array contains the RGB color value corresponding to the color name in the aszColor[] array with the same index. Notice that the RGB macro is used to fabricate RGB colors. This macro is defined in Win32 API header files.

- iSave, iCurrent and iDefault

  These variables are all integers. They contain an item index in the color selection list box. Since they are initialized to zero, on the first call of ColorDlgProc(), they point to the same color - the black color. Remember the GetGMessage() function also uses the black color to initialize the color field of the input GMESSAGE data structure.

  iDefault always points to the black color.

  iCurrent keeps track of the current selection in the color list box.

  iSave is used to save the value of iCurrent on each entry of the color dialog box.

On receipt of the WM_INITDIALOG message, TextDlgProc() initializes the list box with the 16 color names stored in aszColor[] and selects the color using iCurrent as an item index in the list box. It also saves the value of iCurrent into iSave.

When a color is selected, the list box notifies the color dialog box with the WM_COMMAND->LBN_SELCHANGE message. Then ColorDlgProc() will store the item index of the currently selected color in the list box into iCurrent and change the color of the static control with the selected color.

The color change of the static control is achieved in an indirect way. At first the ChangeColor() function is called to create a brush of the selected color and invalidate the static control window. Before repainting the static control window, Windows sends the WM_CTLCOLORSTATIC message to ask the brush to paint the static control window. At this time, ColorDlgProc() returns the handle of the just created brush.

On receipt of the pushbutton command WM_COMMAND->UNDO, ColorDlgProc() restores the initial color using iSave as the item index in the color list box.

On receipt of the pushbutton command WM_COMMAND->DEFAULT, ColorDlgProc() restores the default using iDefault as the item index in the color list box.

On receipt of the WM_DESTROY message, ColorDlgProc() retrieves the color value in the aulColor[] array using iCurrent as an index and places it in the color field of the input GMESSAGE data structure.

```
// depend.c for Win32
#include <windows.h>
#include <commctrl.h>
#include <string.h>

#include "depend.h"
#include "resource.h"

// Static global variables
extern HINSTANCE hInst;
static PGMESSAGE pgmsg;
static HWND hwndText, hwndColor;
static CHAR szDefault[] = "Hello, the World!";
int iSave = 0, iCurrent = 0, iDefault = 0;
CHAR aszColor[16][16] = {
  "Black", "Dark Blue", "Dark Green", "Dark Cyan", "Dark Red", "Dark Magenta", "Dark Yellow", "Dark Gra
  "White", "Blue",     "Green",     "Cyan",     "Red",     "Magenta",     "Yellow",     "Gray"
};
static ULONG aulColor[] = {
   RGB(    0,    0,    0 ), // black
   RGB(    0,    0, 0x7F ), // dark blue
   RGB(    0, 0x7F,    0 ), // dark green
   RGB(    0, 0x7F, 0x7F ), // dark cyan
   RGB( 0x7F,    0,    0 ), // dark red
   RGB( 0x7F,    0, 0x7F ), // dark magenta
   RGB( 0x7F, 0x7F,    0 ), // dark yellow
   RGB( 0x3F, 0x3F, 0x3F ), // dark gray
   RGB( 0x7F, 0x7F, 0x7F ), // gray
   RGB(    0,    0, 0xFF ), // blue
   RGB(    0, 0xFF,    0 ), // green
   RGB(    0, 0xFF, 0xFF ), // cyan
   RGB( 0xFF,    0,    0 ), // red
   RGB( 0xFF,    0, 0xFF ), // magenta
   RGB( 0xFF, 0xFF,    0 ), // yellow
   RGB( 0xFF, 0xFF, 0xFF ), // White
};

// Private function prototypes
BOOL CALLBACK BookDlgProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK TextDlgProc (HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);
BOOL CALLBACK ColorDlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam);
HWND InitBook ( HWND hwnd );
HWND InitColor( HWND hwnd );
BOOL ChangePage( LPNMHDR pnmhdr );
HBRUSH ChangeColor( HWND hwnd, HBRUSH hOldBrush, ULONG ulNewColor );
```

Figure 141 (Part 1 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

```
// Get greating message default text and color, and initialize common controls
BOOL GetGMessage( PGMESSAGE pgmessage )
{
   // Initialize the common controls.
   InitCommonControls();

   strcpy( pgmessage->szText, szDefault );
   pgmessage->ulColor = aulColor[iDefault];

   return TRUE;
}

BOOL SetGMessage( HWND hwnd, PGMESSAGE pgmessage )
{
   pgmsg = pgmessage;
   DialogBox( hInst, MAKEINTRESOURCE(IDD_BOOK), hwnd, BookDlgProc );

   return TRUE;
}

// Book Dialog procedure.                                          */
BOOL CALLBACK BookDlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   switch (msg)
   {
      case WM_INITDIALOG:
         InitBook( hwnd );
         break;

      case WM_NOTIFY:
         ChangePage( (LPNMHDR)lParam );
         return TRUE;

      case WM_CLOSE:
      {
         EndDialog( hwnd, TRUE );
         return TRUE;
      }
   }

   return FALSE;
}

// Text Dialog procedure.                                          */
BOOL CALLBACK TextDlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
   static HWND hwndEdit;
```

Figure 141 (Part 2 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

```
    switch (msg)
    {
        case WM_INITDIALOG:
            hwndEdit = GetDlgItem( hwnd, IDC_EDIT_TEXT );
            SetWindowText( hwndEdit, pgmsg->szText );
            break;

        case WM_DESTROY:
        {
            GetWindowText( hwndEdit, pgmsg->szText, sizeof pgmsg->szText );
            EndDialog( hwnd, TRUE );
            return TRUE;
        }
        case WM_COMMAND:
            switch( LOWORD(wParam) )
            {
                case IDC_UNDO:
                    SetWindowText( hwndEdit, pgmsg->szText );
                    break;

                case IDC_DEFAULT:
                    SetWindowText( hwndEdit, szDefault );
                    break;

                default:
                    break;
            } /* endswitch */
            break;
    }

    return FALSE;
}

// Color Dialog procedure
BOOL CALLBACK ColorDlgProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    static HWND hwndStatic, hwndList;
    static HBRUSH hBrush;

    switch (msg)
    {
        case WM_INITDIALOG:
            iSave = iCurrent;
            hwndStatic = GetDlgItem( hwnd, IDC_STATIC_COLOR );
            hBrush = ChangeColor( hwndStatic, NULL, aulColor[iCurrent] );
            hwndList = InitColor( hwnd );
            break;
```

Figure 141 (Part 3 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

```
      case WM_DESTROY:
         pgmsg->ulColor = aulColor[iCurrent];
         return TRUE;

      case WM_COMMAND:
         switch( LOWORD(wParam) )
         {
            case IDC_UNDO:
               iCurrent = iSave;
               SendMessage( hwndList, LB_SETCURSEL, (WPARAM)iCurrent, 0 );
               hBrush = ChangeColor( hwndStatic, hBrush, aulColor[iCurrent] );
               break;

            case IDC_DEFAULT:
               iCurrent = iDefault;
               SendMessage( hwndList, LB_SETCURSEL, (WPARAM)iCurrent, 0 );
               hBrush = ChangeColor( hwndStatic, hBrush, aulColor[iCurrent] );
               break;

            case IDC_LIST_COLOR:
               if ( HIWORD(wParam) == LBN_SELCHANGE )
               {
                  iCurrent = SendMessage( (HWND) lParam, LB_GETCURSEL, 0, 0 );
                  hBrush = ChangeColor( hwndStatic, hBrush, aulColor[iCurrent] );
               }
               break;
         } /* endswitch */
         break;

      case WM_CTLCOLORSTATIC:
         if ( (HWND)lParam == hwndStatic )
            return (BOOL)hBrush;
         break;
   }

   return FALSE;
}

// Initialize note book
HWND InitBook( HWND hwnd )
{
   HWND hwndBook;
   TC_ITEM tci;

   // Get TabControl handle
   hwndBook = GetDlgItem( hwnd, IDC_NOTEBOOK );

   // Create the Text and Color dialogs
   hwndText  = CreateDialog( hInst, MAKEINTRESOURCE(IDD_TEXT),  hwndBook, TextDlgProc );
   hwndColor = CreateDialog( hInst, MAKEINTRESOURCE(IDD_COLOR), hwndBook, ColorDlgProc );
```

Figure 141 (Part 4 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

```
    // Insert Text Page
    tci.mask   = TCIF_TEXT;
    tci.iImage = -1;
    tci.pszText= "&Text";
    TabCtrl_InsertItem( hwndBook, 0 , &tci );

    // Insert Color Page
    tci.mask   = TCIF_TEXT;
    tci.iImage = -1;
    tci.pszText= "&Color";
    TabCtrl_InsertItem( hwndBook, 1 , &tci );

    ShowWindow( hwndText, SW_SHOW );

    return hwndBook;
}

HWND InitColor( HWND hwnd )
{
    HWND hwndList;
    int i;

    hwndList = GetDlgItem( hwnd, IDC_LIST_COLOR );
    for ( i = 0; i < 16; i++ )
    {
        // Set Red slider range
        SendMessage( hwndList, LB_ADDSTRING, 0, (LPARAM)aszColor[i] );
    }
    SendMessage( hwndList, LB_SETCURSEL, (WPARAM)iCurrent, 0 );

    return hwndList;
}

// Change note book page
BOOL ChangePage( LPNMHDR pnmhdr )
{
    if( pnmhdr->code == TCN_SELCHANGE )
    {
        int iTab = TabCtrl_GetCurSel( (HWND)pnmhdr->hwndFrom );

        ShowWindow( hwndText,  (iTab == 0) ? SW_SHOW : SW_HIDE );
        ShowWindow( hwndColor, (iTab == 1) ? SW_SHOW : SW_HIDE );
    }
    return TRUE;
}

HBRUSH ChangeColor( HWND hwnd, HBRUSH hOldBrush, ULONG ulNewColor )
{
    HBRUSH hNewBrush;
```

Figure 141 (Part 5 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

```
    if ( hOldBrush )
       DeleteObject( hOldBrush );
    hNewBrush = CreateSolidBrush( aulColor[iCurrent] );
    InvalidateRect( hwnd, NULL, TRUE );

    return hNewBrush;
}
```

Figure 141 (Part 6 of 6). Mixed Mode Win32 Platform Specific Code (DEPEND.C)

5.5 Migration

Since the mixed mode sample program is a mixed mode application, the migration steps are different from the ones of the Open32 applications described in Chapter 3, “Howdy, World!” on page 79 and Chapter 4, “MDI Sample Program” on page 103. However many of the steps for migrating Open32 applications can still be applied here.

5.5.1 Converting Resources

Follow the migration steps described in 3.1, “Overview of the Migration Process” on page 79 to convert the MIXMODE.RC resource file.

Since the Win32 Tab control in the message property dialog box is not supported by Developer API Extensions, SMART cannot translate it and leaves it unchanged. We choose to replace it by its OS/2 equivalent control - the notebook control. Figure 142 shows the converted MIXMODE.RC after modification.

After the resource conversion, you can use the OS/2 dialog box editor to check the message property dialog box, adjust the notebook's size and change its styles and attributes.

```
// mixmode.rc
#include <os2.h>
#include "resource.h"

// Icon
ICON            IDI_ICON   DISCARDABLE    "mixmode.ico"
ICON            IDI_DAPIE  DISCARDABLE    "dapie.ico"
```

Figure 142 (Part 1 of 3). Mixed Mode Converted Resources for OS/2

```
// Menu
MENU IDR_MENU DISCARDABLE
BEGIN
   SUBMENU " File", 0xF200
   BEGIN
      MENUITEM " Properties",              IDM_PROP
      MENUITEM " Exit",                   IDM_EXIT
   END
   SUBMENU " Help", 0xF201
   BEGIN
      MENUITEM " About...",                IDM_ABOUT
   END
END

// Dialog
DLGTEMPLATE IDD_BOOK LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "Message Properties", IDD_BOOK, 12, 2, 257, 125, , FCF_SYSMENU |
         FCF_TITLEBAR
   BEGIN
      NOTEBOOK       IDC_BOOK, 0, 0, 257, 125, BKS_BACKPAGESTR |
                  BKS_MAJORTABTOP | BKS_ROUNDEDTABS |
                  BKS_SPIRALBIND | WS_GROUP
   END
END

DLGTEMPLATE IDD_ABOUT LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "About Mixed Mode Sample", IDD_ABOUT, 59, 44, 233, 95,
         FS_SCREENALIGN | WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
   BEGIN
      DEFPUSHBUTTON   "OK", DID_OK, 84, 7, 63, 14
      ICON          IDI_ICON, IDI_ICON, 200, 11, 20, 16
      CTEXT          "DAPIE Mixmode Sample", IDC_STATIC, 68, 69, 98, 8,
                  DT_WORDBREAK | DT_MNEMONIC
      ICON          IDI_DAPIE, IDC_STATIC, 9, 11, 20, 16
      CTEXT          "(C) Copyright IBM Corp. 1996", IDC_STATIC, 59, 50,
                  115, 8, DT_WORDBREAK | DT_MNEMONIC
      CTEXT          "Developed by IBM ITSC Austin", IDC_STATIC, 52, 31,
                  129, 8, DT_WORDBREAK | DT_MNEMONIC
   END
END
```

Figure 142 (Part 2 of 3). Mixed Mode Converted Resources for OS/2

```
DLGTEMPLATE IDD_TEXT LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "", IDD_TEXT, 3, 0, 233, 94, NOT FS_DLGBORDER
   BEGIN
      DEFPUSHBUTTON   " Undo", IDC_UNDO, 9, 7, 63, 14
      PUSHBUTTON     " Default", IDC_DEFAULT, 85, 7, 63, 14
      LTEXT         "Text:", IDC_STATIC_TEXT, 9, 54, 22, 8, DT_WORDBREAK |
                   DT_MNEMONIC
      ENTRYFIELD     "", IDC_EDIT_TEXT, 11, 39, 211, 11, ES_MARGIN
   END
END

DLGTEMPLATE IDD_COLOR LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "", IDD_COLOR, 3, 0, 233, 94, NOT FS_DLGBORDER
   BEGIN
      PUSHBUTTON    " Undo", IDC_UNDO, 9, 7, 63, 14
      PUSHBUTTON    " Default", IDC_DEFAULT, 79, 7, 63, 14
      LISTBOX      IDC_LIST_COLOR,   9,  24, 132,  61, LS_NOADJUSTPOS | WS_TABSTOP
      LTEXT        "", IDC_STATIC_COLOR, 149,  24,  72,  61, SS_TEXT | DT_WORDBREAK | DT_MNEMONIC
   END
END
```

Figure 142 (Part 3 of 3). Mixed Mode Converted Resources for OS/2

5.5.2 Converting Common Source Code

The change in the resource file does not have any impact on the common source code, because, as explained earlier, the common source code uses the interface functions which isolates it from the platform specific features. You perform the same steps as were described in 4.6, “Migration” on page 129 to convert the MIXMODE.C file.

5.5.3 Converting Platform Specific Source Code

For Windows application developers who are not familiar with OS/2 native API and window classes/messages, the simplest way to convert the platform specific source code is to use the SMART migration tool. Following are the steps for migrating the mixed mode sample DEPEND.C Win32 code to OS/2 code:

1. Defining an LST file

   SMART requires that you build a file with an extension of LST to indicate the names of your source code and header files to be analyzed. The following instructions tell you how to create a LST file.

   a. Go to the **File** pull-down menu and select the **Maintain a List-of-Files...** option, as shown in Figure 143 on page 152.

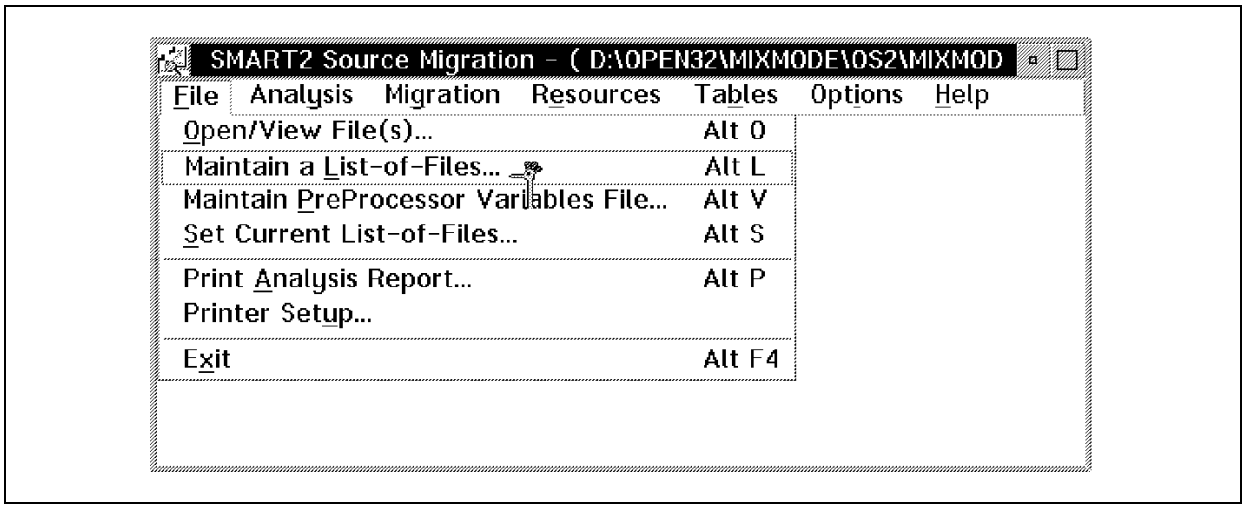


Figure 143. SMART Defined List of Files: File Pull-Down Menu

b. On the Select File for List-of-Files dialog, type the name of the file you wish to create in the File entry field. Select the **OK** pushbutton, as shown in Figure 144 on page 153.

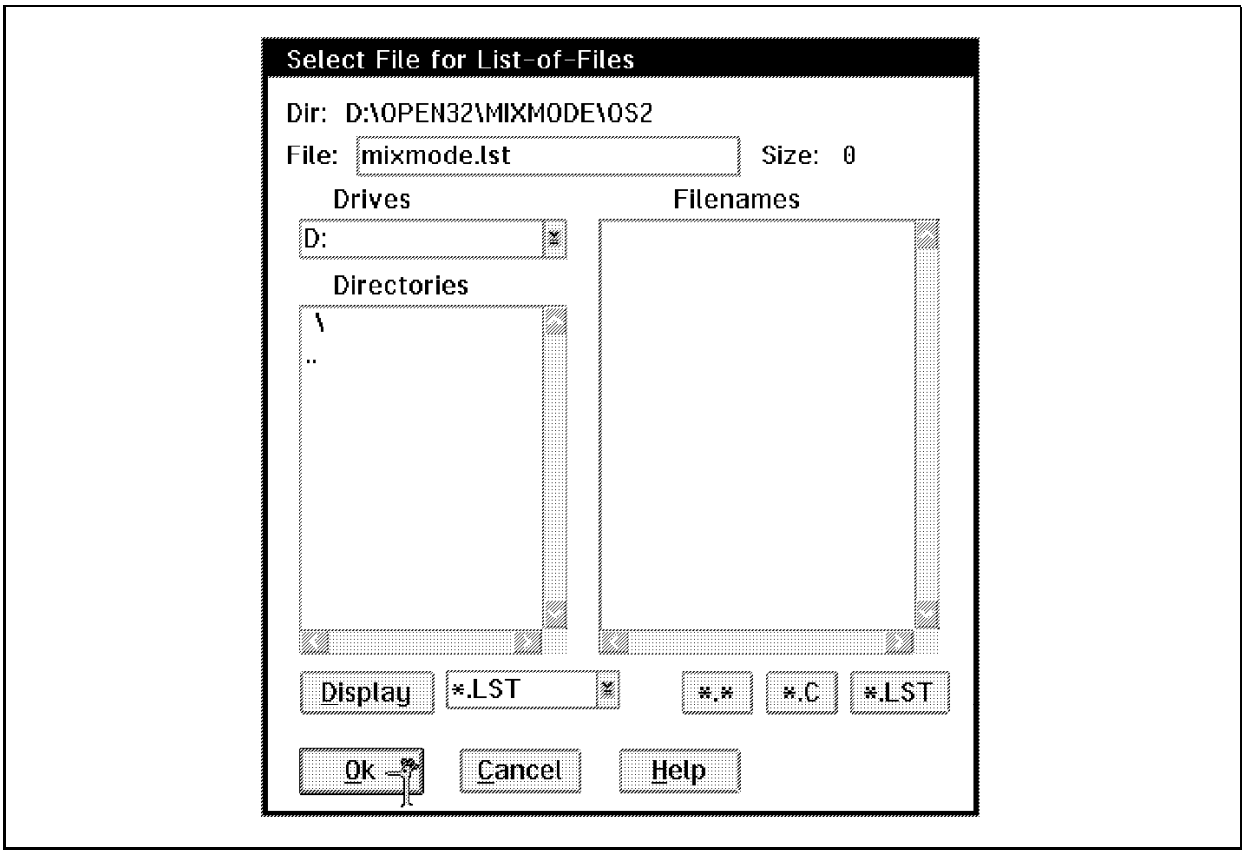


Figure 144. SMART Defined List of Files: Select File for List-of-Files Dialog

c. SMART displays a message box to ask you to confirm the new list file creation. Select the **OK** pushbutton.

d. On the Files List dialog, select the **Add...** pushbutton, as shown in Figure 145 on page 154.

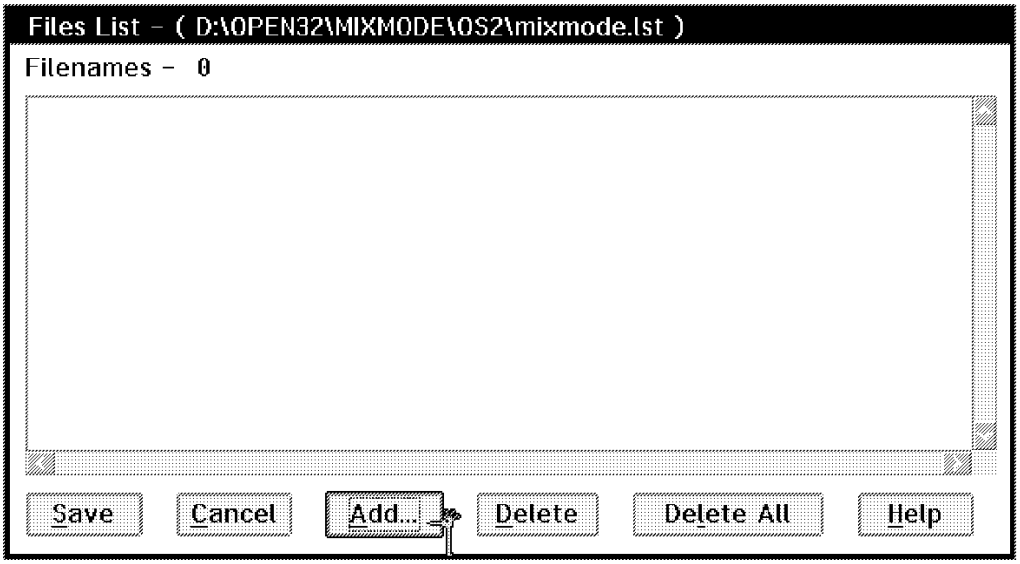


Figure 145. SMART Defined List of Files: Files List Dialog

e. You will be presented the Add Files To List dialog as shown in Figure 146 on page 155. Select the DEPEND.H and DEPEND.C files you want to analyze from the Filenames list box then select the **Add** pushbutton. When you have finished adding files, select the **Close** pushbutton.

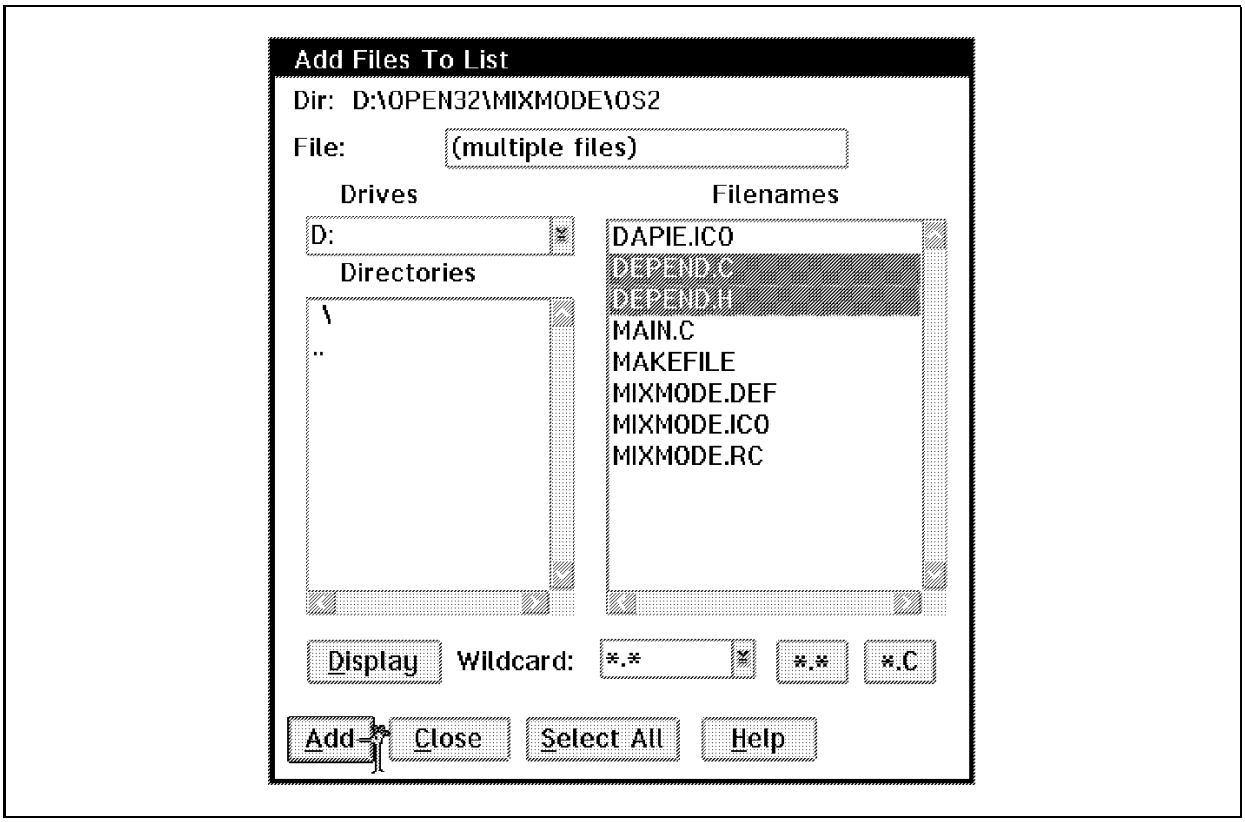


Figure 146. SMART Defined List of Files: Add Files To List Dialog

f. You will be returned to the Files List dialog shown in Figure 145 on page 154. Select the **Save** pushbutton to save the LST file you have created.

2. Selecting the Migration Table

   You need to identify the migration table to be used by SMART for the migration of Win32 code to OS/2 code in the following way:

   a. Go to the Tables pull-down menu and select the **Select Tables...** option, as shown in Figure 147 on page 156.

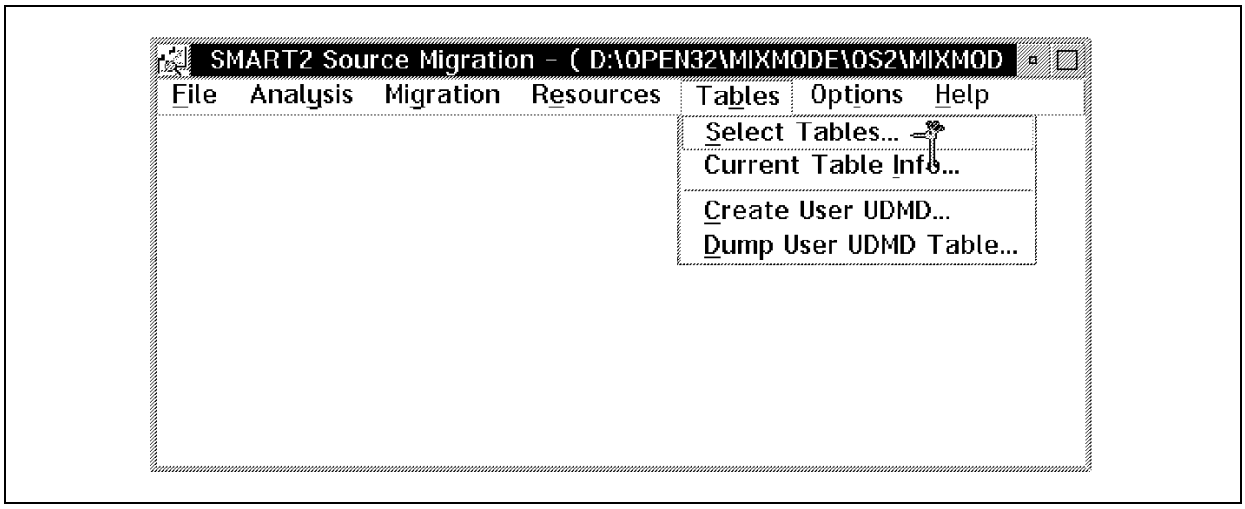


Figure 147. SMART Select Migration Table: Table Pull-Down Menu

b. You will be presented the Migration Tables dialog shown in Figure 148 on page 157. Use the **Select1...** pushbutton to select the SMART TABLES WIN32OS2 WIN32OS2.TBL as the migration table. Make sure the **Do Not Use UDMD Table** check box is checked, then select the **Set** pushbutton to have your setting remembere d by the SMART tool.

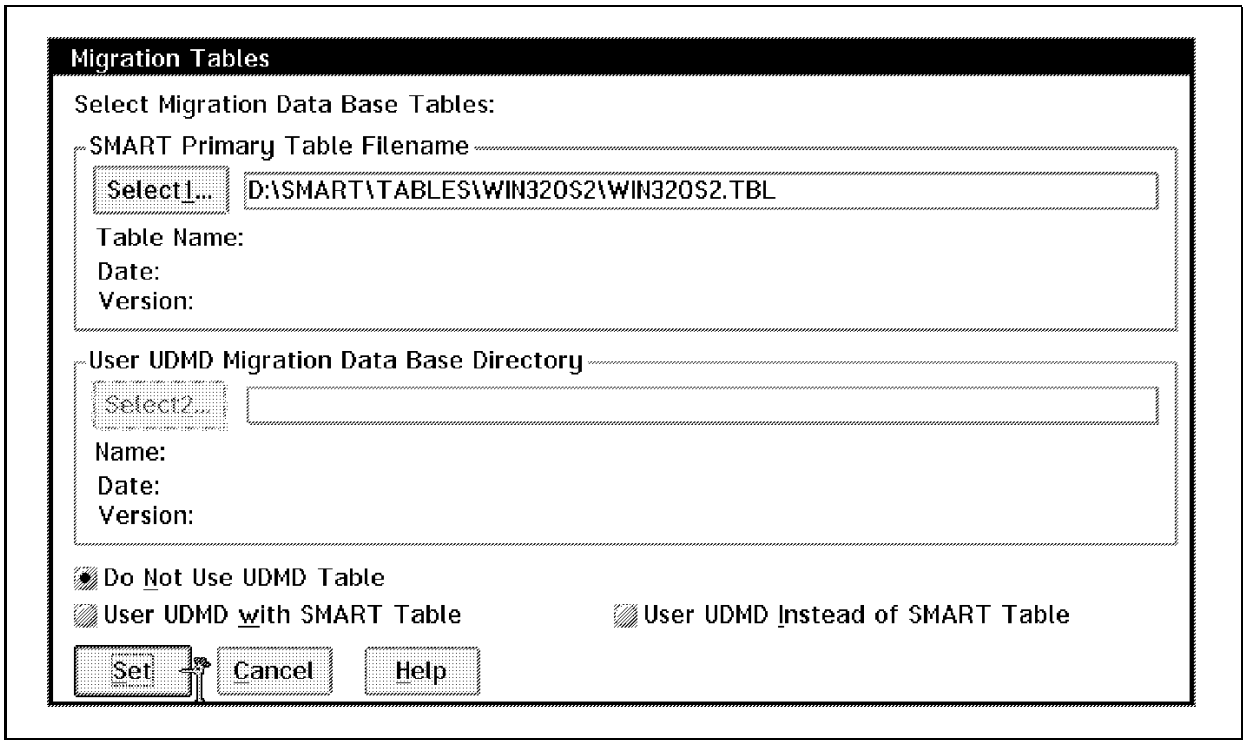


Figure 148. SMART Selected Migration Table: Migration Tables Dialog

3. Analyzing the Source Code

   a. Go to the Analysis pull-down menu and select the **Analyze Source Code...** option, as shown in Figure 149.

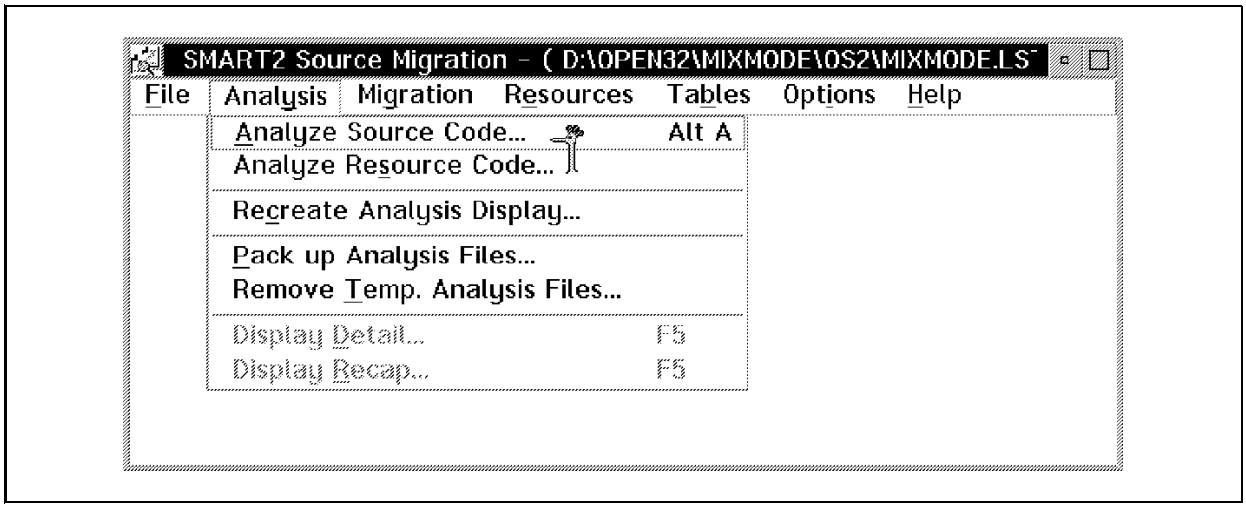


Figure 149. SMART Analyzer Source Code: Analysis Pull-Down Menu

b. You will be presented the Source Code Analysis dialog shown in Figure 150 on page 158. Select the **Process** pushbutton to start the analysis of the files that are placed in the List-of-Files using the migration table you selected.

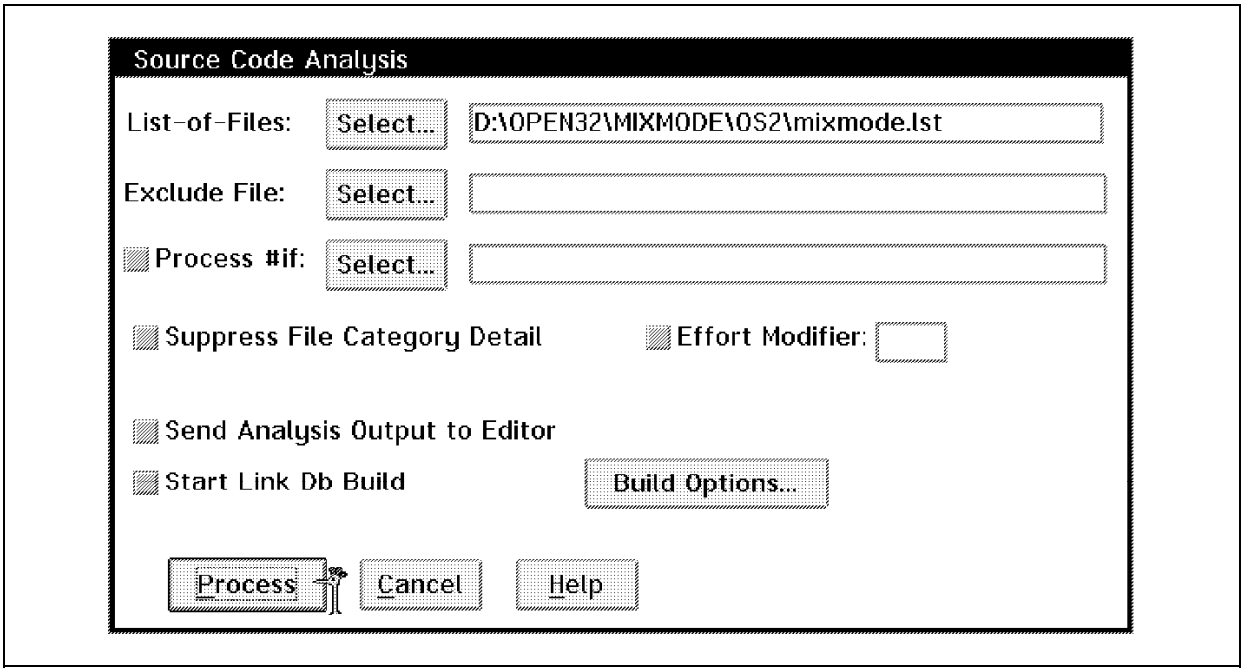


Figure 150. SMART Analyzer Source Code: Source Code Analysis Dialog

c. When the analysis is complete, you will be presented an analysis report as shown in Figure 151 on page 159. It will give you an estimation of the effort required to migrate the source code between Win32 and OS/2.

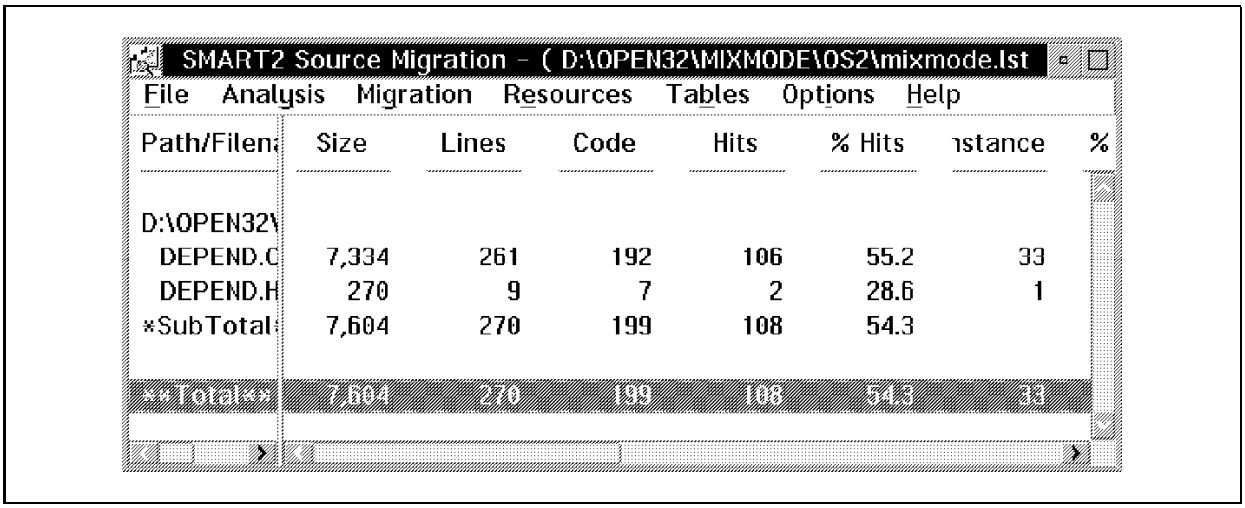


Figure 151. SMART Analyzer Source Code: Source Code Analysis Report

4. Migrating the Source Code

   a. Go to the Migration pull-down menu and select the **Migrate Source Code...** option, as shown in Figure 152.

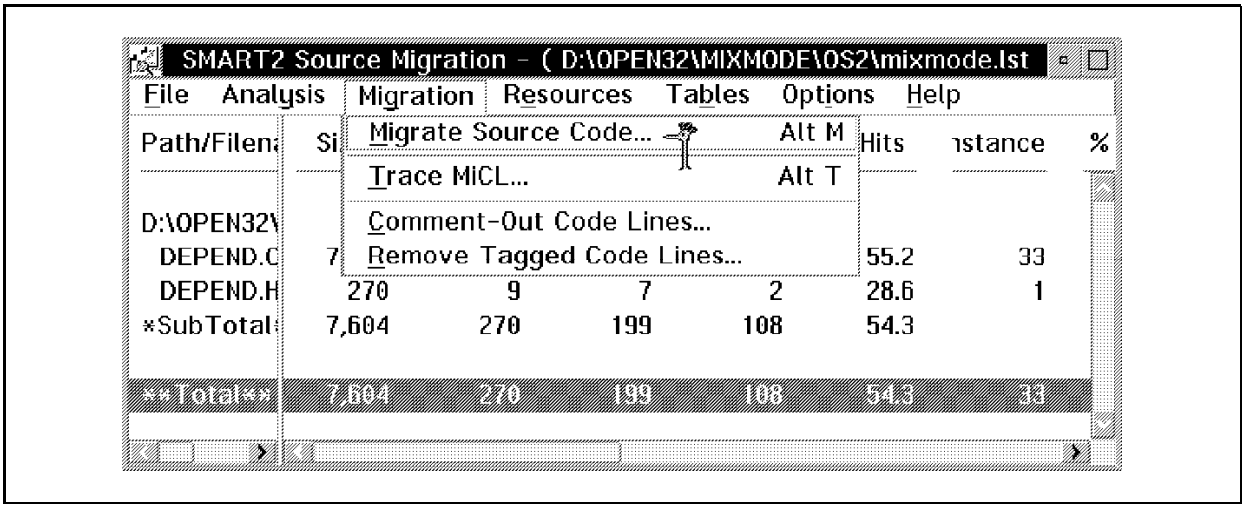


Figure 152. SMART Migrate Source Code: Migrate Pull-Down Menu

   b. You will be presented with the Migration Process Options dialog shown in Figure 153 on page 160. You can use the default values and select the **Process** pushbutton to start the migration.

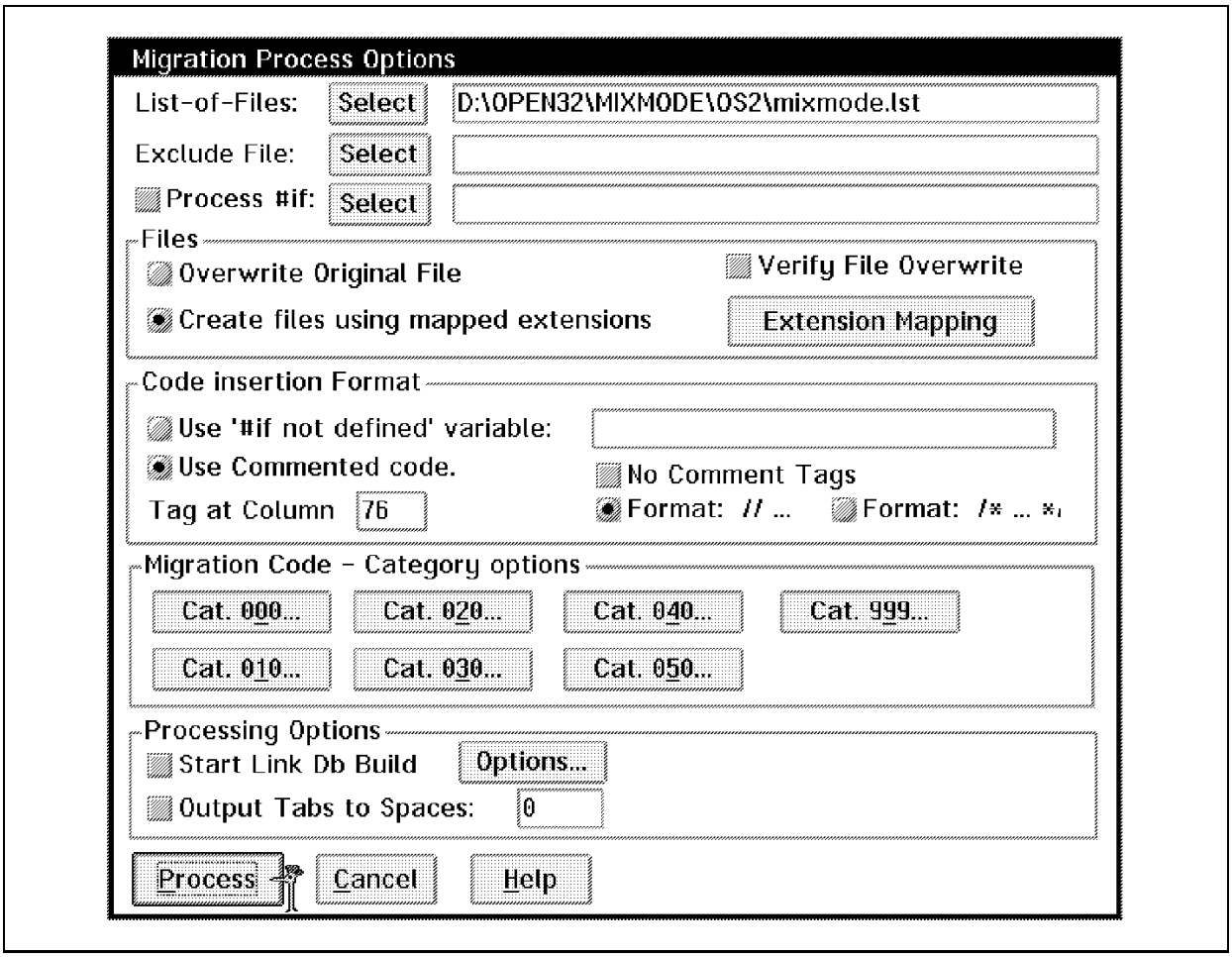


Figure 153. SMART Migrate Source Code: Migration Process Options

c. After the automated SMART migration process is complete, you will need to edit the converted DEPEND.C file to manually complete the migration. You need to replace the Win32 API functions with the OS/2 API functions.

In many cases, SMART is able to suggest equivalent OS/2 API functions or window messages for Win32 ones in the code. The migration work becomes a simple replacement of function names with eventual modification in parameters. You can see how this was done for the DEPEND.C source code by comparing the two versions of the code found in the OS/2 and WIN32 subdirectories of the MIXMODE directory on the CD-ROM in this redbook. You may also visually compare the Win32 version shown in Figure 141 on page 144 with the OS/2 version shown in Figure 154 on page 162.

There are some functions where a one-to-one correspondence does not exist. Following are the differences between Win32 and OS/2 that were also addressed in the DEPEND.C code sample.

- Notification messages

  On OS/2, the notification messages are sent through the WM_CONTROL message instead of the WM_COMMAND message on Win32. For example, the list box LBN_SELCHANGE notification message in the color dialog box is sent through the WM_COMMAND message on the Win32 version. When migrated to OS/2, it must not only be changed to LN_SELECT, but also put in the WM_CONTROL processing section.

- OS/2 Notebook Control verses Win32 Tab Control

  OS/2 notebook control switches the pages for you when page tab is clicked while Win32 Tab control sends you a notification message. In the Win32 code, you hide the undesired page and show the desired one. This code is not needed in the OS/2 migrated code.

- Static control color

  In OS/2 you call WinSetPresParam() to change a window's presentation parameters including its colors. In Win32 for the static control, you have to create a brush of the color you wish to use and provide the handle of the brush to Windows each time it repaints the static control.

- RGB Color

  Although both OS/2 and Win32 use a ULONG to represent a RGB color, the format used is different as shown in Table 5.

Table 5. RGB Color Format on OS/2 and Win32

Platform	**Bit 0-7**	**Bit 8-15**	**Bit 16-23**
OS/2	Blue	Green	Red
Win32	Red	Green	Blue

You need to use the RGBCONVERT macro to convert the color format before returning it back to the common source code.

```
//******************************************************
//* Depend.c for OS/2
//******************************************************
#define INCL_DOS
#define INCL_WIN
#define INCL_GPI
#include <os2.h>
#include <string.h>

#include "depend.h"
#include "resource.h"

// Macro definition
#define RGB(r,g,b) ( (ULONG)(((UCHAR)(b)|((USHORT)((UCHAR)(g))<<8))|(((USHORT)(UCHAR)(r))<<16)))
#define RGBCONVERT(rgb) ( (ULONG)(((rgb & 0x000000FF)<<16)|((rgb & 0x00FF0000)>>16)|(rgb & 0x0000FF00)

// Private variable definition
static PGMESSAGE pgmsg;
static CHAR szDefault[] = "Hello, the World!";
int iSave = 0, iCurrent = 0, iDefault = 0;
CHAR aszColor[16][16] = {
  "Black", "Dark Blue", "Dark Green", "Dark Cyan", "Dark Red", "Dark Magenta", "Dark Yellow", "Dark Gra
  "Gray",  "Blue",     "Green",     "Cyan",     "Red",     "Magenta",     "Yellow",     "White"
};
static ULONG aulColor[] = {
   RGB(   0,   0,   0 ), // black
   RGB(   0,   0, 0x7F ), // dark blue
   RGB(   0, 0x7F,   0 ), // dark green
   RGB(   0, 0x7F, 0x7F ), // dark cyan
   RGB( 0x7F,   0,   0 ), // dark red
   RGB( 0x7F,   0, 0x7F ), // dark magenta
   RGB( 0x7F, 0x7F,   0 ), // dark yellow
   RGB( 0x3F, 0x3F, 0x3F ), // dark gray
   RGB( 0x7F, 0x7F, 0x7F ), // gray
   RGB(   0,   0, 0xFF ), // blue
   RGB(   0, 0xFF,   0 ), // green
   RGB(   0, 0xFF, 0xFF ), // cyan
   RGB( 0xFF,   0,   0 ), // red
   RGB( 0xFF,   0, 0xFF ), // magenta
   RGB( 0xFF, 0xFF,   0 ), // yellow
   RGB( 0xFF, 0xFF, 0xFF ), // White
};

MRESULT EXPENTRY BookDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
MRESULT EXPENTRY TextDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
MRESULT EXPENTRY ColorDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
HWND InitBook( HWND hwnd );
HWND InitColor( HWND hwnd );
ULONG AddPage( HWND hwndBook, HWND hwndDlg, PSZ pszTabText );
BOOL ChangeColor( HWND hwnd, ULONG ulColor );
```

Figure 154 (Part 1 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

```
// Get greeting message default text and color, and initialize common controls
BOOL GetGMessage( PGMESSAGE pgmessage )
{
  strcpy( pgmessage->szText, szDefault );
  pgmessage->ulColor = RGBCONVERT(aulColor[iDefault]);

  return TRUE;
}

BOOL SetGMessage( HWND hwnd, PGMESSAGE pgmessage )
{
   pgmsg = pgmessage;

   WinDlgBox( HWND_DESKTOP,
            hwnd,        /* handle of the owner       */
            BookDlgProc,  /* dialog procedure address   */
            NULLHANDLE,   /* location of dialog resource */
            IDD_BOOK,    /* resource identifier        */
            NULL );      /* application-specific data   */
  return TRUE;
}

// Book Dialog procedure.                                          */
MRESULT EXPENTRY BookDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    switch (msg)
    {
        case WM_INITDLG:
           InitBook( hwnd );
           break;

        case WM_COMMAND:
           break;

        case WM_CONTROL:
           break;

        default:
           return WinDefDlgProc (hwnd, msg, mp1, mp2);
    }

    return FALSE;
}

// Text Dialog procedure.
MRESULT EXPENTRY TextDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
    static HWND hwndEdit;
```

Figure 154 (Part 2 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

```
    switch (msg)
    {
       case WM_INITDLG:
          hwndEdit = WinWindowFromID( hwnd, IDC_EDIT_TEXT );
          WinSetWindowText( hwndEdit, pgmsg->szText );
          break;

       case WM_DESTROY:
          WinQueryWindowText( hwndEdit, sizeof pgmsg->szText, pgmsg->szText );
          break;

       case WM_COMMAND:
          switch( SHORT1FROMMP( mp1 ) )
          {
             case IDC_UNDO:
                WinSetWindowText( hwndEdit, pgmsg->szText );
                break;

             case IDC_DEFAULT:
                WinSetWindowText( hwndEdit, szDefault );
                break;
          } /* endswitch */
          break;

       default:
          return WinDefDlgProc( hwnd, msg, mp1, mp2 );
    }

    return FALSE;
}

MRESULT EXPENTRY ColorDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
  static HWND hwndList, hwndStatic;

    switch (msg)
    {
       case WM_INITDLG:
          iSave = iCurrent;
          hwndStatic = WinWindowFromID( hwnd, IDC_STATIC_COLOR );
          hwndList = InitColor( hwnd );
          break;

       case WM_DESTROY:
       {
          ULONG ulColor = aulColor[iCurrent];
          pgmsg->ulColor = RGBCONVERT(ulColor);
          break;
       }
```

Figure 154 (Part 3 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

```
      case WM_COMMAND:
         switch( SHORT1FROMMP( mp1 ) )
         {
            case IDC_UNDO:
               WinSendMsg(hwndList, LM_SELECTITEM,
                     MPFROMLONG(iSave), (MPARAM)TRUE );
               break;

            case IDC_DEFAULT:
               WinSendMsg( hwndList, LM_SELECTITEM,
                     MPFROMLONG(iDefault), (MPARAM)TRUE );
               break;
         } /* endswitch */
         break;

      case WM_CONTROL:
         if (SHORT2FROMMP(mp1) == LN_SELECT )
         {
            LONG lIndex = WinQueryLboxSelectedItem( (HWND) mp2 );
            if ( lIndex != LIT_NONE ) {
               iCurrent = lIndex;
               ChangeColor( hwndStatic, aulColor[iCurrent] );
            } /* endif */
         } /* endif */
         break;

      default:
         return WinDefDlgProc (hwnd, msg, mp1, mp2);
   }

   return FALSE;
}

// Initialize note book
HWND InitBook( HWND hwnd )
{
   HWND  hwndBook, hwndDlg;
   RECTL rect;

   // Get Note Book Window Handle.
   hwndBook = WinWindowFromID( hwnd, IDC_BOOK );
```

Figure 154 (Part 4 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

```
    // Set NoteBook background page color
    WinSendMsg( hwndBook, BKM_SETNOTEBOOKCOLORS,
            (MPARAM)SYSCLR_BUTTONMIDDLE,
            (MPARAM)BKA_BACKGROUNDPAGECOLOR );

    // Set NoteBook major tab dimension
    WinSendMsg( hwndBook, BKM_SETDIMENSIONS,
            MPFROM2SHORT( 100, 30 ),
            MPFROMSHORT( BKA_MAJORTAB ) );

    // Get Client Rect Size.
    WinQueryWindowRect( hwnd, &rect );

    // Load Text dialog box
    hwndDlg = WinLoadDlg( hwndBook,
                hwndBook,
                TextDlgProc,
                0,
                IDD_TEXT,
                NULL );

    // Add Text dialog page to the Notebook
    AddPage( hwndBook, hwndDlg, " Text" );

    // Load Color dialog box
    hwndDlg = WinLoadDlg( hwndBook,
                hwndBook,
                ColorDlgProc,
                0,
                IDD_COLOR,
                NULL );

    // Add Color dialog page to the Notebook
    AddPage( hwndBook, hwndDlg, " Color" );

    return hwndBook;
}

HWND InitColor( HWND hwnd )
{
    HWND   hwndList;
    ULONG  i;

    hwndList = WinWindowFromID( hwnd, IDC_LIST_COLOR );

    // Insert color items into the list box
    for ( i = 0; i < 16; i++) {
       WinInsertLboxItem( hwndList, i, aszColor[i] );
    } /* endfor */
```

Figure 154 (Part 5 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

```
    // Set color selection
    WinSendMsg( hwndList, LM_SELECTITEM, (MPARAM)iCurrent, (MPARAM)TRUE );

    return hwndList;
}

// Add a dialog to a Notebook
ULONG AddPage( HWND hwndBook, HWND hwndDlg, PSZ pszTabText )
{
   ULONG ulPageId, rc;

   // Insert New Page into NoteBook.
   ulPageId = (ULONG)WinSendMsg( hwndBook, BKM_INSERTPAGE,
                         NULL,
                         MPFROM2SHORT(BKA_AUTOPAGESIZE|BKA_MAJOR,BKA_FIRST) );

   // Set page tab text
   WinSendMsg( hwndBook, BKM_SETTABTEXT,
             (MPARAM)ulPageId,
             MPFROMP(pszTabText) );

   //* Associate window with the inserted notebook page.              */
   WinSendMsg( hwndBook,
           BKM_SETPAGEWINDOWHWND,
           MPFROMLONG(ulPageId),
           MPFROMHWND(hwndDlg));

   return ulPageId;
}

BOOL ChangeColor( HWND hwnd, ULONG ulNewColor )
{
    RGB2 rgb;

    rgb.bBlue    =  ulNewColor & 0x000000FF;
    rgb.bGreen   = (ulNewColor & 0x0000FF00) >> 8;
    rgb.bRed     = (ulNewColor & 0x00FF0000) >> 16;
    rgb.fcOptions = 0;

    WinSetPresParam( hwnd, PP_BACKGROUNDCOLOR, sizeof rgb, &rgb );

    return TRUE;
}
```

Figure 154 (Part 6 of 6). Mixed Mode OS/2 Platform Specific Code (DEPEND.C)

5.6 Application Enhancement on OS/2

Once you have completed the migration of the OS/2 version of the mixed mode sample program, you may wish to enhance it beyond the Win32 version using features unique to OS/2. This section describes how to enhance the user interface concerning the color selection.

In the Win32 version, the colors are presented by a list box in text mode as shown in Figure 136 on page 136. You cannot see the corresponding color before selecting it.

A better solution is to use the OS/2 ValueSet control for color choice presentation and selection. The ValueSet control provides an easy-to-use WYSIWYG way of user-machine interaction. You see all the available colors before choosing one. Win32 has no equivalent control class to OS/2. Adding it to the OS/2 version of the mixed mode sample program is an enhancement which cannot then be ported back to the Win32 version of the sample.

For the mixed mode sample program porting, we found the ValueSet programming was easier than porting the list box. You do not need to intercept the notification messages of color selection change. You just query the color selection at the end of the color dialog.

To do this enhancement, you need to make some changes in the resource file (MIXMODE.RC) and the platform specific source code file (DEPEND.C). In the MIXMODE.RC resource file, you need to replace, in the color dialog box, the list box and the static control by a ValueSet control. If you are not familiar with OS/2 ValueSet control resource script, you can use OS/2 dialog editor to define a ValueSet control. Figure 155 shows the enhanced version of the MIXMODE.RC resource file with the changes made for the color dialog box. This enhanced resource file can be compared to the originally migrated OS/2 resource file shown in Figure 142 on page 149.

```
#include <os2.h>
#include "resource.h"

// Icon
ICON            IDI_ICON    DISCARDABLE     "mixmode.ico"
ICON            IDI_DAPIE   DISCARDABLE     "dapie.ico"
```

Figure 155 (Part 1 of 3). Mixed Mode Enhanced Resources for OS/2 (MIXMODE.RC)

```
// Menu
MENU IDR_MENU DISCARDABLE
BEGIN
   SUBMENU " File", 0xF200
   BEGIN
      MENUITEM " Properties",              IDM_PROP
      MENUITEM " Exit",                   IDM_EXIT
   END
   SUBMENU " Help", 0xF201
   BEGIN
      MENUITEM " About...",                IDM_ABOUT
   END
END

// Dialog
DLGTEMPLATE IDD_BOOK LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "Message Properties", IDD_BOOK, 12, 2, 257, 125, , FCF_SYSMENU |
          FCF_TITLEBAR
   BEGIN
      NOTEBOOK       IDC_BOOK, 0, 0, 257, 125, BKS_BACKPAGESTR |
                    BKS_MAJORTABTOP | BKS_ROUNDEDTABS |
                    BKS_SPIRALBIND | WS_GROUP
   END
END

DLGTEMPLATE IDD_ABOUT LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "About Mixed Mode Sample", IDD_ABOUT, 59, 44, 233, 95,
          FS_SCREENALIGN | WS_VISIBLE, FCF_SYSMENU | FCF_TITLEBAR
   BEGIN
      DEFPUSHBUTTON   "OK", DID_OK, 84, 7, 63, 14
      ICON          IDI_ICON, IDI_ICON, 200, 11, 20, 16
      CTEXT         "DAPIE Mixed Mode Sample", IDC_STATIC, 68, 69, 98, 8,
                    DT_WORDBREAK | DT_MNEMONIC
      ICON          IDI_DAPIE, IDC_STATIC, 9, 11, 20, 16
      CTEXT         "(C) Copyright IBM Corp. 1996", IDC_STATIC, 59, 50,
                    115, 8, DT_WORDBREAK | DT_MNEMONIC
      CTEXT         "Developed by IBM ITSC Austin", IDC_STATIC, 52, 31,
                    129, 8, DT_WORDBREAK | DT_MNEMONIC
   END
END
```

Figure 155 (Part 2 of 3). Mixed Mode Enhanced Resources for OS/2 (MIXMODE.RC)

```
DLGTEMPLATE IDD_COLOR LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "", IDD_COLOR, 3, 0, 233, 94, NOT FS_DLGBORDER
   BEGIN
      PUSHBUTTON     " Undo", IDC_UNDO, 9, 7, 63, 14
      PUSHBUTTON     " Default", IDC_DEFAULT, 79, 7, 63, 14
      VALUESET       IDC_LIST_COLOR, 28, 32, 166, 52, VS_RGB | VS_BORDER |
                     WS_GROUP
                     CTLDATA 8, 0, 2, 8
   END
END

DLGTEMPLATE IDD_TEXT LOADONCALL MOVEABLE DISCARDABLE
BEGIN
   DIALOG  "", IDD_TEXT, 3, 0, 233, 94, NOT FS_DLGBORDER
   BEGIN
      DEFPUSHBUTTON   " Undo", IDC_UNDO, 9, 7, 63, 14
      PUSHBUTTON     " Default", IDC_DEFAULT, 85, 7, 63, 14
      LTEXT          "Text:", IDC_STATIC_TEXT, 9, 54, 22, 8, DT_WORDBREAK |
                     DT_MNEMONIC
      ENTRYFIELD     "", IDC_EDIT_TEXT, 11, 39, 211, 11, ES_MARGIN
   END
END
```

Figure 155 (Part 3 of 3). Mixed Mode Enhanced Resources for OS/2 (MIXMODE.RC)

The DEPEND.C source code file needs to be modified to support the use of the ValueSet control as follows:

- Static control is no longer needed
- Global variable iCurrent is no longer needed
- List box related messages need to be replaced by the equivalent of the ValueSet control
- Variables iSave and iDefault change into ULONG to contain the row number and column number of an item in the ValueSet control instead of an index to list box items

Figure 156 on page 171 shows the enhanced version of the color page. The source code file for the enhanced OS/2 version of the dependent part of the mixed mode application can be found in ENHOS2 subdirectory of the OS2 subdirectory of the MIXMODE directory on the CD-ROM included in this redbook. In Figure 157 on page 171 a copy of the enhanced version of DEPEND.C is shown if you wish to compare it to either the Win32 version in Figure 141 on page 144 or the migrated OS/2 version shown in Figure 154 on page 162.

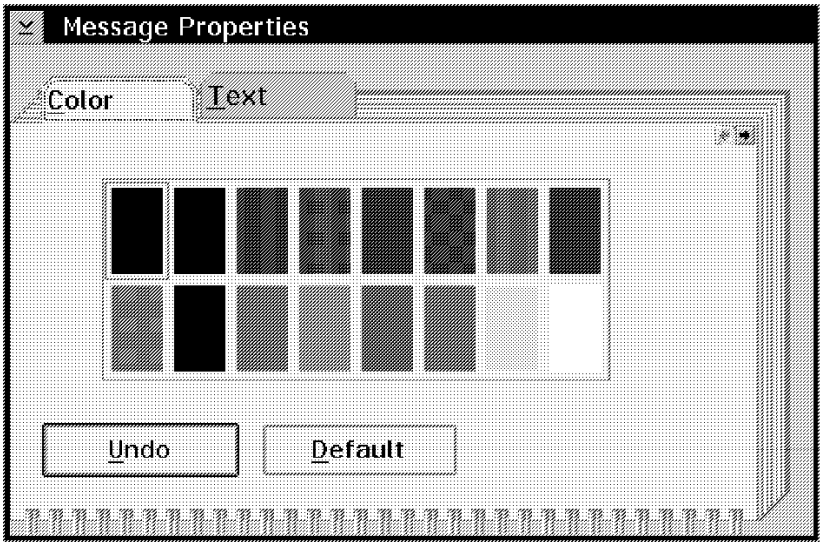


Figure 156. Mixed Mode Enhanced Color Page on OS/2

```
//******************************************************
//* Depend.c for OS/2
//******************************************************
#define INCL_DOS
#define INCL_WIN
#include <os2.h>
#include <string.h>

#include "depend.h"
#include "resource.h"

// Macro definition
#define RGB(r,g,b) ( (ULONG)(((UCHAR)(b)|((USHORT)((UCHAR)(g))<<8))|(((USHORT)(UCHAR)(r))<<16)))
#define RGBCONVERT(rgb) ( (ULONG)(((rgb & 0x000000FF)<<16)|((rgb & 0x00FF0000)>>16)|(rgb & 0x0000F
```

Figure 157 (Part 1 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
// Private variable definition
static PGMESSAGE pgmsg;
static CHAR szDefault[] = "Hello, the World!";
static ULONG iSave = 0x00010001, iDefault = 0x00010001; // (Row,Col) = (1,1)
static ULONG aulColor[] = {
   RGB(   0,   0,   0 ), // black
   RGB(   0,   0, 0x7F ), // dark blue
   RGB(   0, 0x7F,   0 ), // dark green
   RGB(   0, 0x7F, 0x7F ), // dark cyan
   RGB( 0x7F,   0,   0 ), // dark red
   RGB( 0x7F,   0, 0x7F ), // dark magenta
   RGB( 0x7F, 0x7F,   0 ), // dark yellow
   RGB( 0x3F, 0x3F, 0x3F ), // dark gray
   RGB( 0x7F, 0x7F, 0x7F ), // gray
   RGB(   0,   0, 0xFF ), // blue
   RGB(   0, 0xFF,   0 ), // green
   RGB(   0, 0xFF, 0xFF ), // cyan
   RGB( 0xFF,   0,   0 ), // red
   RGB( 0xFF,   0, 0xFF ), // magenta
   RGB( 0xFF, 0xFF,   0 ), // yellow
   RGB( 0xFF, 0xFF, 0xFF ), // White
};

MRESULT EXPENTRY BookDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
MRESULT EXPENTRY TextDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
MRESULT EXPENTRY ColorDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
HWND InitBook( HWND hwnd );
HWND InitColor( HWND hwnd );
ULONG AddPage( HWND hwndBook, HWND hwndDlg, PSZ pszTabText );

// Get greeting message default text and color, and initialize common controls
BOOL GetGMessage( PGMESSAGE pgmessage )
{
  strcpy( pgmessage->szText, szDefault );
  pgmessage->ulColor = 0;

  return TRUE;
}
```

Figure 157 (Part 2 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
BOOL SetGMessage( HWND hwnd, PGMESSAGE pgmessage )
{
   pgmsg = pgmessage;

   WinDlgBox( HWND_DESKTOP,
            hwnd,        /* handle of the owner       */
            BookDlgProc,  /* dialog procedure address    */
            NULLHANDLE,   /* location of dialog resource */
            IDD_BOOK,     /* resource identifier        */
            NULL );      /* application-specific data   */

  return TRUE;
}

// Book Dialog procedure.                                          */
MRESULT EXPENTRY BookDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   switch (msg)
   {
      case WM_INITDLG:
         InitBook( hwnd );
         break;

      case WM_COMMAND:
         break;

      case WM_CONTROL:
         break;

      default:
         return WinDefDlgProc (hwnd, msg, mp1, mp2);
   }

   return FALSE;
}

// Text Dialog procedure.
MRESULT EXPENTRY TextDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   static HWND hwndEdit;

   switch (msg)
   {
      case WM_INITDLG:
         hwndEdit = WinWindowFromID( hwnd, IDC_EDIT_TEXT );
         WinSetWindowText( hwndEdit, pgmsg->szText );
         break;
```

Figure 157 (Part 3 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
      case WM_DESTROY:
         WinQueryWindowText( hwndEdit, sizeof pgmsg->szText, pgmsg->szText );
         break;

      case WM_COMMAND:
         switch( SHORT1FROMMP( mp1 ) )
         {
            case IDC_UNDO:
               WinSetWindowText( hwndEdit, pgmsg->szText );
               break;

            case IDC_DEFAULT:
               WinSetWindowText( hwndEdit, szDefault );
               break;
         } /* endswitch */
         break;

      default:
         return WinDefDlgProc( hwnd, msg, mp1, mp2 );
   }

   return FALSE;
}

MRESULT EXPENTRY ColorDlgProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
  static HWND hwndList;

   switch (msg)
   {
      case WM_INITDLG:
         hwndList = InitColor( hwnd );
         break;

      case WM_DESTROY:
      {
         ULONG ulColor;
         iSave = (ULONG)WinSendMsg( hwndList, VM_QUERYSELECTEDITEM,
                        NULL, NULL );
         ulColor = (ULONG)WinSendMsg( hwndList, VM_QUERYITEM,
                MPFROMLONG(iSave), NULL );
         pgmsg->ulColor = RGBCONVERT(ulColor);
         break;
      }
```

Figure 157 (Part 4 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
       case WM_COMMAND:
          switch( SHORT1FROMMP( mp1 ) )
          {
             case IDC_UNDO:
                WinSendMsg(hwndList, VM_SELECTITEM,
                       MPFROMLONG(iSave), NULL );
                break;

             case IDC_DEFAULT:
                WinSendMsg( hwndList, VM_SELECTITEM,
                       MPFROMLONG(iDefault), NULL );
                break;
          } /* endswitch */
          break;

       default:
          return WinDefDlgProc (hwnd, msg, mp1, mp2);
   }

   return FALSE;
}

// Initialize note book
HWND InitBook( HWND hwnd )
{
   HWND  hwndBook, hwndDlg;
   RECTL rect;

   // Get Note Book Window Handle.
   hwndBook = WinWindowFromID( hwnd, IDC_BOOK );

   // Set NoteBook background page color
   WinSendMsg( hwndBook, BKM_SETNOTEBOOKCOLORS,
           (MPARAM)SYSCLR_BUTTONMIDDLE,
           (MPARAM)BKA_BACKGROUNDPAGECOLOR );

   // Set NoteBook major tab dimension
   WinSendMsg( hwndBook, BKM_SETDIMENSIONS,
           MPFROM2SHORT( 100, 30 ),
           MPFROMSHORT( BKA_MAJORTAB ) );

   // Get Client Rect Size.
   WinQueryWindowRect( hwnd, &rect );

   // Load Text dialog box
   hwndDlg = WinLoadDlg( hwndBook,
                hwndBook,
                TextDlgProc,
                0,
                IDD_TEXT,
                NULL );
```

Figure 157 (Part 5 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
    // Add Text dialog page to the Notebook
    AddPage( hwndBook, hwndDlg, " Text" );

    // Load Color dialog box
    hwndDlg = WinLoadDlg( hwndBook,
                hwndBook,
                ColorDlgProc,
                0,
                IDD_COLOR,
                NULL );

    // Add Color dialog page to the Notebook
    AddPage( hwndBook, hwndDlg, " Color" );

    return hwndBook;
}

HWND InitColor( HWND hwnd )
{
    HWND   hwndList;
    USHORT usRow, usCol;
    ULONG  i;

    hwndList = WinWindowFromID( hwnd, IDC_LIST_COLOR );
    // Set the color value for each item in each row and column.        */
    for ( usRow = 1, i = 0; usRow < 3; usRow++ ) {
        for ( usCol = 1; usCol < 9; usCol++ ) {
            WinSendMsg( hwndList,           /* Value set window handle    */
                VM_SETITEM,              /* Message for setting items  */
                MPFROM2SHORT(usRow,usCol),  /* Set item in row, column    */
                MPFROMLONG(aulColor[i]));   /* to the color red.         */
            i++;
        } /* endfor */
    } /* endfor */

    // Set selection
    WinSendMsg( hwndList, VM_SELECTITEM, MPFROMLONG(iSave), NULL );

    return hwndList;
}
```

Figure 157 (Part 6 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

```
// Add a dialog to a Notebook
ULONG AddPage( HWND hwndBook, HWND hwndDlg, PSZ pszTabText )
{
   ULONG ulPageId, rc;

   // Insert New Page into NoteBook.
   ulPageId = (ULONG)WinSendMsg( hwndBook, BKM_INSERTPAGE,
                         NULL,
                         MPFROM2SHORT(BKA_AUTOPAGESIZE|BKA_MAJOR,BKA_FIRST) );
   // Set page tab text
   WinSendMsg( hwndBook, BKM_SETTABTEXT,
             (MPARAM)ulPageId,
             MPFROMP(pszTabText) );

   //* Associate window with the inserted notebook page.              */
   WinSendMsg( hwndBook,
           BKM_SETPAGEWINDOWHWND,
           MPFROMLONG(ulPageId),
           MPFROMHWND(hwndDlg));

   return ulPageId;
}
```

Figure 157 (Part 7 of 7). Mixed Mode Enhanced Platform Specific Code for OS/2 (DEPEND.C)

Chapter 6. Named Pipe Sample Program

In Chapter 5, “Mixed Mode Sample Program” on page 133 you learned a technique to resolve the problem with unsupported window classes. This technique works but we lose the single source code file for our program. As a result, extra effort is required to migrate the Win32 application to the OS/2 environment. Also the maintenance of the platform dependent code on both OS/2 and Win32 platform will mean extra work.

This chapter will discuss a technique which allows for the use of Win32 APIs not supported by Open32. Using this technique, we are able to compile and execute the same common source file on both the OS/2 and Windows systems without making changes to the source file. This technique uses an interface layer that handles the unsupported Win32 API calls and uses equivalent OS/2 functions to perform the requests from the common source code application.

In this chapter we discuss the development of the named pipe interface layer along with a sample program that uses the Win32 named pipe API call which the interface layer supports. Also included in this chapter are discussions on:

- How to develop both named pipe server and client applications
- How to develop multiple thread applications
- How to use event semaphores for inter-thread communications

6.1 Application's Overview

The named pipe sample program is made up of two applications: one is a named pipe server application and the other is a named pipe client application. They communicate to each other through a named pipe on a local workstation or on a LAN.

6.1.1 Named Pipe Server Application's Overview

In the named pipe server application, you use the **Open** option of the File pull-down menu, as shown in Figure 158 on page 180, to create a named pipe instance.

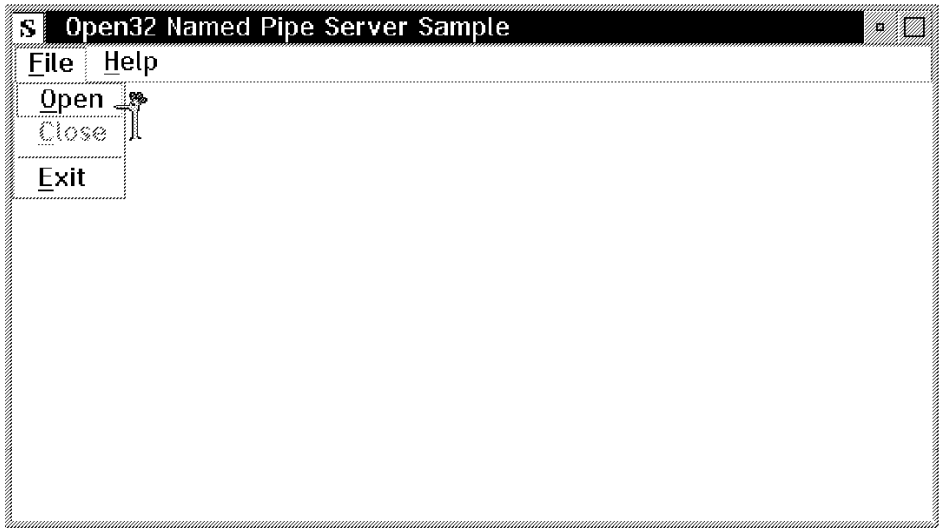


Figure 158. Named Pipe Server: File Pull-Down

Each named pipe instance is visually represented by an MDI child window, as shown in Figure 159.

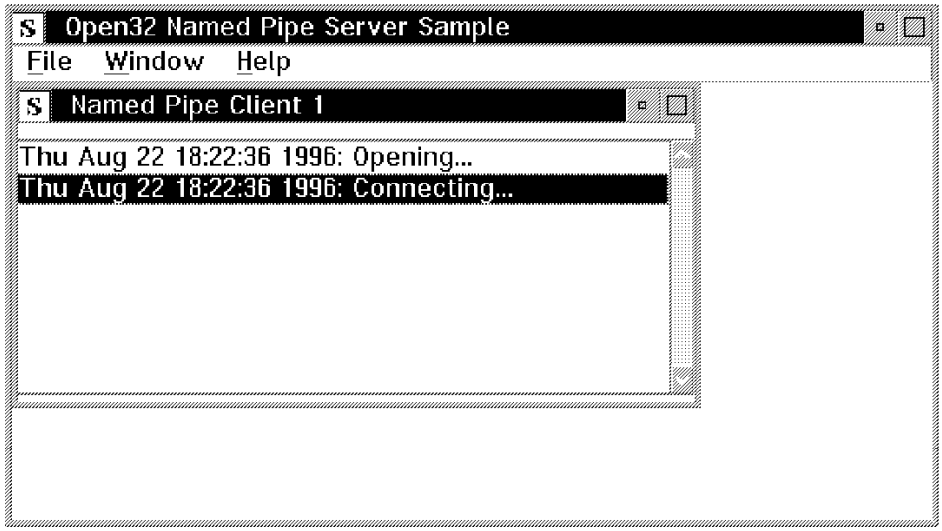


Figure 159. Named Pipe Server: Named Pipe Instance Window

Once the named pipe instance is connected, you can exchange data with the client through the named pipe. This can be done by using the **Send** option of the File pull-down menu to open a dialog box, type the message

text and send it to the connected client by selecting the **OK** button, as shown in Figure 160 on page 181.

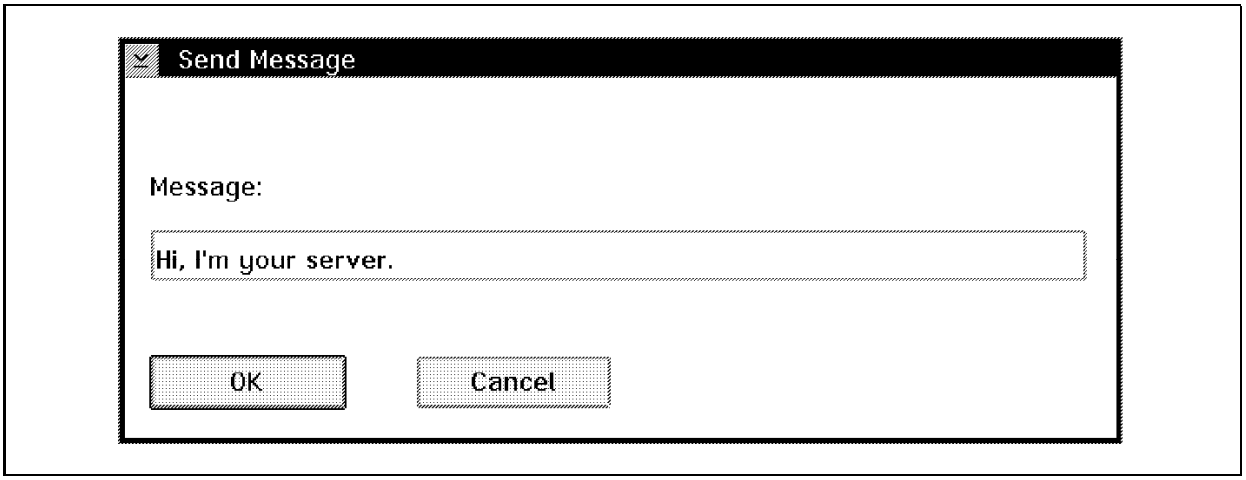


Figure 160. Named Pipe Server: Send Message Dialog

You receive a message by using the **Receive** option of the File pull-down menu. All the messages sent or received will be displayed in the named pipe instance's window.

When you finish the dialog, you can use the **Close** option using the File pull-down menu to close the named pipe instance. The corresponding named pipe instance's window will be closed and a close message will be sent to the client to alert it.

The named pipe server application is similar to the MDI sample and used the same logic to manage MDI child windows. In the named pipe sample these child windows are referred to as named pipe instance's windows. You can use the options of the Window pull-down menu to manipulate these windows in the same way as with the MDI sample program described in 4.2, “User's Interface” on page 105.

Communication errors will be logged in the named pipe server application. For example, if the client closes the named pipe instance before the server does this, a pipe broken message will be logged and the named pipe instance will be closed as well as its window.

6.1.2 Named Pipe Client Application's Overview

In the named pipe client application, you use the **Open** option of the File pull-down menu, as shown in Figure 161 on page 182, to open a named pipe instance.

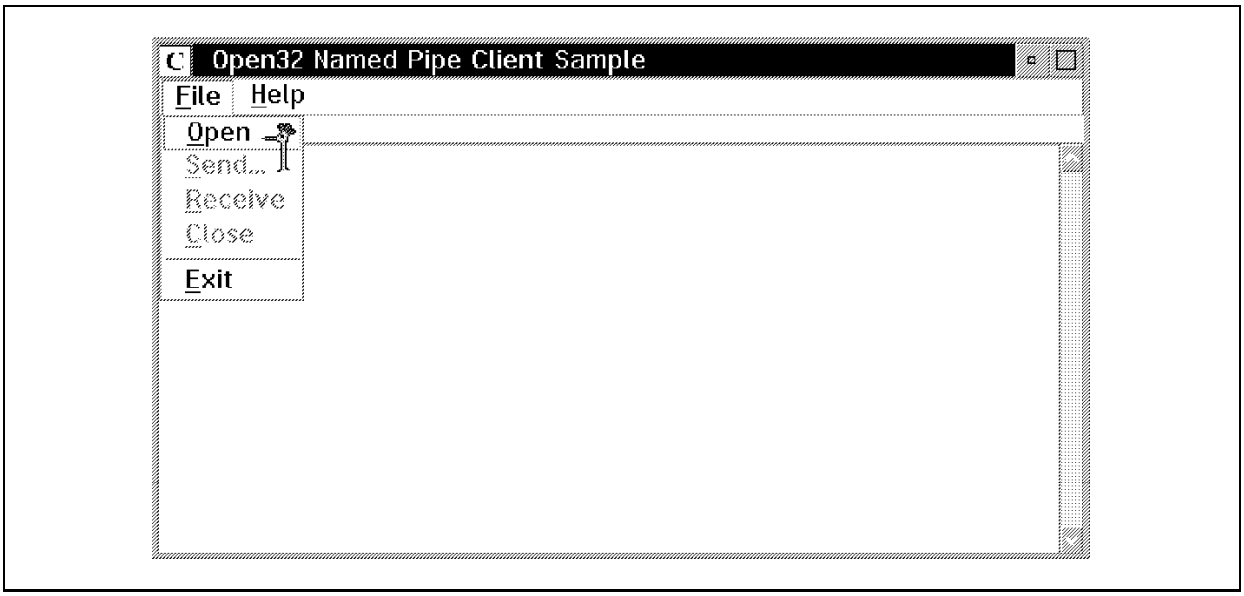


Figure 161. Named Pipe Client: File Pull-Down

Once the named pipe instance is connected, data is exchanged with the named pipe server through a named pipe. This is accomplished using the **Send** option of the File pull-down menu to present the Send Message window where you type the message text before selecting the **OK** button to send the message to the server. Figure 162 shows the Send Message window of the named pipe client sample program.

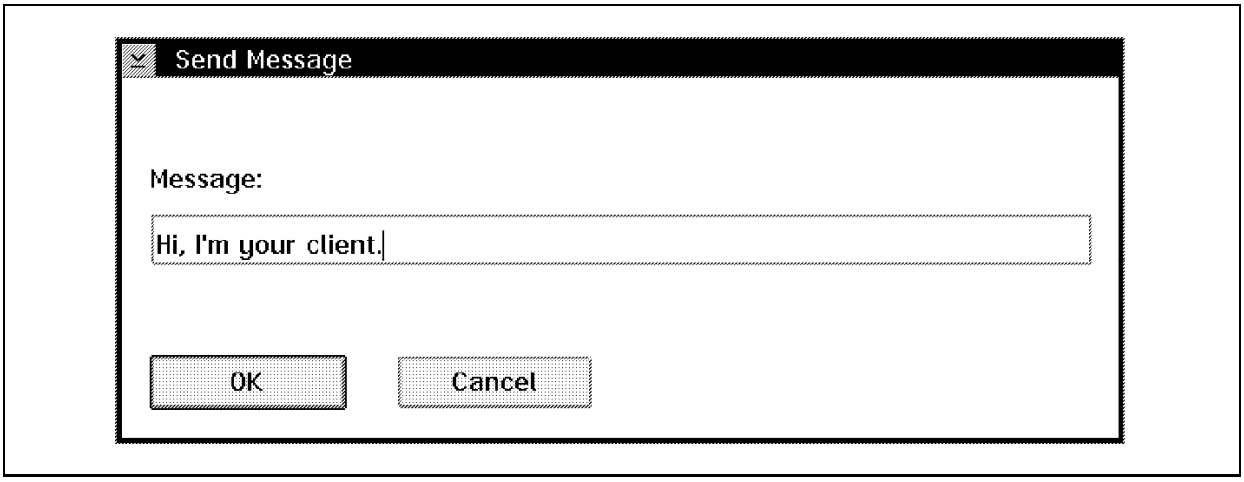


Figure 162. Named Pipe Client: Send Message Dialog

The **Receive** option of the File pull-down menu can be used to receive messages from the server. All the messages sent to or received from the

server are displayed in the main window of the named pipe client application.

When you finish the dialog with the server, you can use the **Close** option of the File pull-down menu to close the named pipe instance. Like the server, the client sends a close message to the server.

Communication errors are handled in the named pipe client application similar to what is done in the the named pipe server application. If the server closes the named pipe instance before the client does, a pipe broken message will be displayed in the main window.

6.2 Source Files

Table 6 lists all the source files used by the named pipe sample program. You will find these files on the CD-ROM supplied in this redbook in the NPIPE directory. The NPIPE has three subdirectories: SERVER, CLIENT and OS2NP. The SERVER subdirectory contains the named pipe server application code while the CLIENT subdirectory contains the named pipe client application code. The OS2NP subdirectory contains the function library for OS/2. Both SERVER and CLIENT subdirectories contain an OS2 and WIN32 subdirectory. The OS2 and WIN32 subdirectories contain, respectively, platform specific files for OS/2 and Windows while the SERVER and CLIENT subdirectories contain the files common to both OS/2 and Windows.

Table 6 (Page 1 of 2). Named Pipe Sample Program Source Files

LOCATION	NAME	DESCRIPTION
NPIPE SERVER	SERVER.C	Server common source code file
NPIPE SERVER	RESOURCE.H	Server common resource header file
NPIPE SERVER OS2	MAIN.C	Server OS/2 specific source file
NPIPE SERVER OS2 and NPIPE SERVER WIN32	SERVER.RC	Server OS/2 and Windows specific resource file
NPIPE SERVER OS2	MAKEFILE	Server OS/2 specific makefile
NPIPE SERVER WIN32	SERVER.MAK	Server Windows specific makefile
NPIPE SERVER OS2	SERVER.DEF	Server OS/2 specific module definition file
NPIPE SERVER OS2 and NPIPE SERVER WIN32	SERVER.ICO	Server OS/2 and Windows specific icon resource file
NPIPE CLIENT OS2	CLIENT.H	OS/2 specific header file
NPIPE CLIENT	CLIENT.C	Client common source code file

Table 6 (Page 2 of 2). Named Pipe Sample Program Source Files

LOCATION	NAME	DESCRIPTION
NPIPE CLIENT	RESOURCE.H	Client common resource header file
NPIPE CLIENT OS2	MAIN.C	Client OS/2 specific source file
NPIPE CLIENT OS2 and NPIPE CLIENT WIN32	CLIENT.RC	Client OS/2 and Windows specific resource file
NPIPE CLIENT OS2	MAKEFILE	Client Windows specific makefile
NPIPE CLIENT WIN32	CLIENT.MAK	Client Windows specific makefile
NPIPE CLIENT OS2	CLEINT.DEF	Client OS/2 specific module definition file
NPIPE CLIENT OS2 and NPIPE CLIENT WIN32	CLIENT.ICO	Client OS/2 and Windows specific icon resource file
NPIPE OS2NP	NPIPE.H	Library OS/2 specific source header file
NPIPE OS2NP	NPIPE.C	Library OS/2 specific source code file
NPIPE OS2NP	MAKEFILE	Library OS/2 specific makefile

6.3 Application Design

Named pipes are a means of interprocess communication where applications use named pipes to exchange data. The data or service provider applications are called named pipe server applications while the data or service consumer applications are called named pipe client applications. Inside each application, there is a user interface component and a data communication component the later which uses named pipes. The user interface component gets data requests (send or receive messages) from the end user and forwards these requests to the data communication component. The data communications component then sends or receives the data request through the named pipe. Figure 163 on page 185 shows the architecture of the named pipe server and client application.

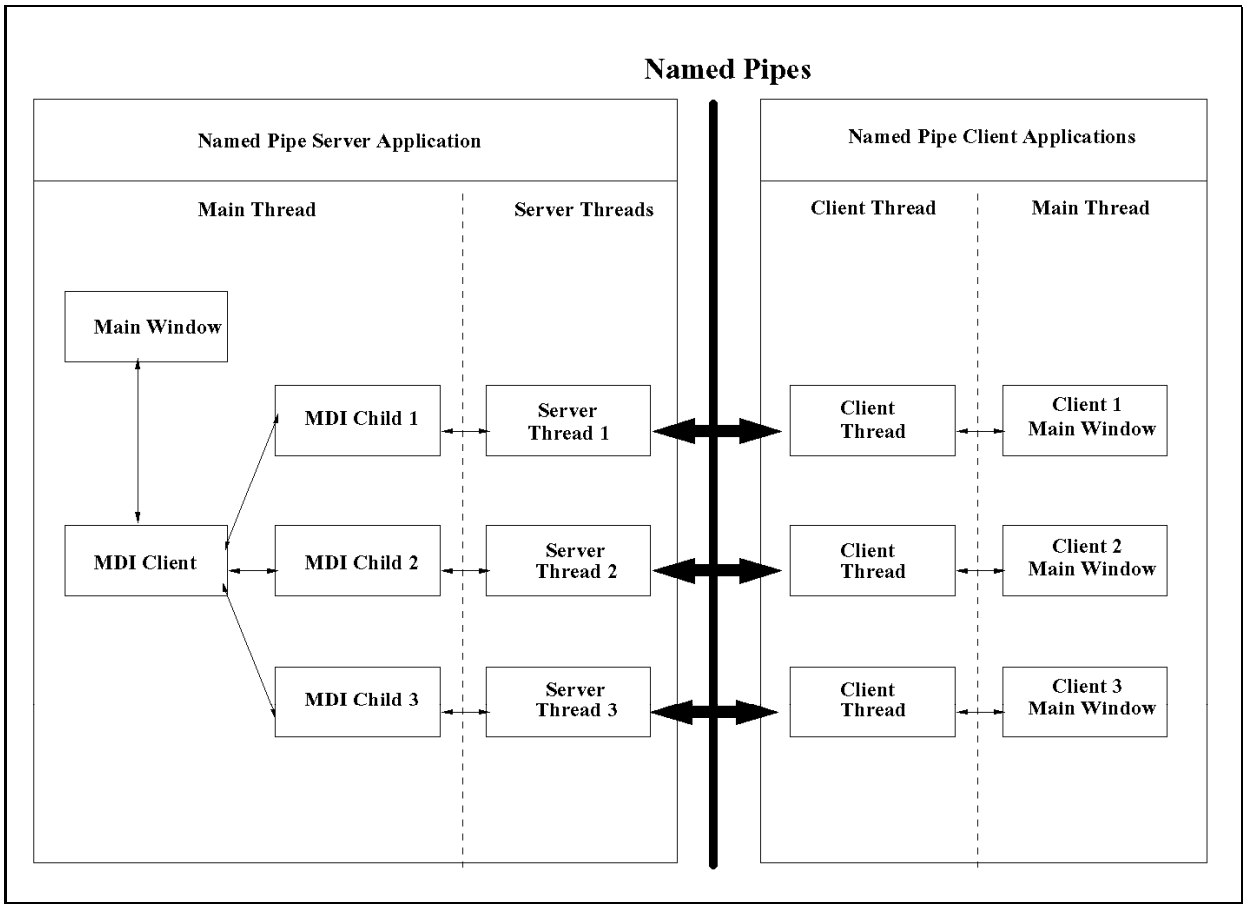


Figure 163. Named Pipe Sample Program Architecture

There are two design points which are implemented in both the named pipe server and client programs. The first is the user interface component is part of the application main thread while the named pipe component is executed in secondary threads called server thread and client thread respectively for the named pipe server and client programs. The reason for this is that actual sending or receiving of data over the named pipe may take a long time hence reducing the responsiveness of the user interface if there was only one thread. However, this separation means that the interthread communication needs to be properly coordinated which leads to the second design consideration.

The second design point is the main thread uses an event semaphore to signal the data communications thread. When the user interface component needs to request data or service of the data communications thread it will block the data communications thread by setting the event semaphore. After setting the event semaphore the user interface component places a request in a shared data structure. The user interface component then

clears the event semaphore to unblock the named pipe thread. Once unblocked, the named pipe thread retrieves the data or service request from the shared data structure and performs the request.

The name of the named pipe for the sample application we selected is Open32. This has been hard-coded into both the server and client programs.

6.4 Coding

The application development starts on the Windows NT platform using Microsoft Visual C++ Version 4.0. Windows 95 does not support Win32 named pipe APIs. Named pipe server applications cannot be developed on Windows 95 machines. However you can still develop named pipe client applications on Windows 95 and run them to access remote named pipes through LANs.

6.4.1 Server Application Coding

The named pipe server application design is based on the MDI sample program. With this design there is always an MDI frame window, and an MDI client window with one or more MDI child windows representing a named pipe instance. In addition, the named pipe server application uses:

- MDI child windows to display messages sent to or received from named pipe clients
- Secondary threads to communicate with the named pipe clients
- Event semaphores to coordinate the MDI child windows and its named pipe server thread

The common source (SERVER.C) for the server program can be found on the CD-ROM in this redbook in the SERVER subdirectory of the NPIPE directory. The resource file (SERVER.RC) can be found in the WIN32 subdirectory of the SERVER directory on the CD-ROM. You may wish to reference these files to see how the application design that follows was actually coded in the program.

In the same way as the MDI sample program, the named pipe server application starts by creating the MDI frame window and the MDI client window in the main thread. Then it goes into the window message loop looking for input from the user.

Upon receipt of the **Open** command from the File pull-down menu, the MDI frame window creates an MDI child window. This new MDI child window does the following things at initialization time:

- Create a list box in its client area

- Create an event semaphore using CreateEvent()
- Create a secondary thread using CreateThread()
- Allocate a data structure of type NPCLIENT

The event semaphore is initialized in the non-signaled state. This will block all threads waiting on it.

The new secondary thread, called named pipe server thread, creates in turn an instance of the named pipe OPEN32 using CreateNamedPipe(). If successful, it notifies its MDI child window and then connects to it waiting for a named pipe client to open it. After the named pipe is connected, the server thread blocks itself on the event semaphore by calling WaitForSingleObject(). Notice there is an MDI child window, an event semaphore and a server thread created for each named pipe instance. The handles of the event semaphore and of the named pipe instance are stored in the NPCLIENT data structure and the address of the data structure is recorded in this MDI child window's data area. The list box will be used to store all the messages sent to or received from the named pipe instance.

Upon receipt of the **Send** command from the File pull-down menu, the main thread opens a dialog box to prompt the user to enter the message to be sent and puts the message text into the NPCLIENT data structure of the currently active MDI child window. It signals the event semaphore of which the handle is stored in the data structure. This unblocks the corresponding server thread. The unblocked server thread retrieves the message text from the NPCLIENT data structure, then writes it to the named pipe. It notifies its MDI child window by posting a message, then blocks itself on the event semaphore again.

Upon receipt of the **Receive** command from the File pull-down menu, the main thread puts the request into the NPCLIENT data structure of the currently active MDI child window then signals the related event semaphore to unblock the corresponding server thread. The unblocked server thread reads the named pipe, writes the message received to the NPCLIENT data structure, notifies the main window by posting a message then blocks itself on the event semaphore again.

Upon receipt of the **Close** command from the File pull-down menu, the main thread puts the request into the NPCLIENT data structure of the currently active MDI child window and signals the related event semaphore to unblock the corresponding server thread. Finally, it closes the MDI child window in question. The unblocked server thread closes the named pipe and the event semaphore, deallocates the NPCLIENT data structure and returns. This ends the thread.

6.4.2 Client Application Coding

The named pipe client program borrows a lot of its logic from the server program. It uses multiple threads to separate the user interface and data communications code. Also, an event semaphore mechanism is used to manage the interthread communication. However, the code is simpler since the client program only has one data communicate section for one named pipe instance.

The common source (CLIENT.C) for the server program can be found on the CD-ROM in this redbook in the CLIENT subdirectory of the NPIPE directory. The resource file (CLIENT.RC) can be found in the WIN32 subdirectory of the SERVER directory on the CD-ROM. You may wish to reference these files to see how the application design that follows was actually coded in the program.

The WinMain() function is like a normal Windows application. It registers the main application window class, creates the main window then enters into the window message loop. At initialization the main window creates an event semaphore for later use for the interthread communication, and a list box to log the messages sent to, or received from, the named pipe as well as the error messages.

Upon receipt of the **Open** command from the File pull-down menu, the main thread creates a secondary thread called the named pipe client thread. This secondary thread opens the named pipe OPEN32, then blocks itself on the event semaphore created at initialization.

Upon receipt of the **Send** command from the File pull-down menu, the main thread opens a dialog box to prompt the user to type a message to send. It puts the message text into the NPCLIENT communication data structure then signals the event semaphore. This unblocks the client thread. The unblocked client thread retrieves the message text from the NPCLIENT data structure, then writes it to the named pipe. It notifies the main window by posting a message, then blocks itself on the event semaphore again.

Upon receipt of the **Receive** command from the File pull-down menu, the main thread puts the request into the NPCLIENT communication data structure, then signals the event semaphore to unblock the client thread. The unblocked client thread reads the named pipe, writes the message received into the NPCLIENT data structure, notifies the main window by posting a message, then blocks itself on the event semaphore again.

Upon receipt of the **Close** command from the File pull-down menu, the main thread puts the request into the NPCLIENT communication data structure and signals the event semaphore to unblock the client thread. The

unblocked client thread closes the named pipe and returns. This ends the thread.

6.5 Migration

Both the named pipe server and client application use window classes not supported by Open32. Simply follow the migration steps described in Chapter 3, “Howdy, World!” on page 79 to convert the resources.

However, both of them use Win32 named pipe APIs which are not currently supported by Open32, but have OS/2 equivalent APIs. They would not be compiled on OS/2 since Open32 header files do not contain the required function prototypes and Open32 libraries do not contain these function references. One solution to this problem is to extract the platform dependent code and group it in a separate source code file, one version for Win32 and another for OS/2. This technique was described in Chapter 5, “Mixed Mode Sample Program” on page 133 to resolve the problem with unsupported window classes. One evident drawback with this technique is that it reduces the common source code base size and complicates the source code maintenance task because of the two versions of the platform dependent source code.

By providing the header file OS2NP.H which defines the Win32 APIs that are being implemented with the technique described in this chapter along with the library file OS2NP.LIB for the function prototypes defined in the header file OS2NP.H the Win32 version of the source code remains intact when migrated to OS/2. You perform the same steps as described in Chapter 3, “Howdy, World!” on page 79 for migrating the source code.

You with need to make a small changes to the MAKEFILE file as shown in Figure 164 on page 190 for the server and Figure 165 on page 190 for the client. Since the named pipe server and client applications are multiple thread applications, the /Gm+ compile option is added in the MAKEFILE files. Also the OS2NP.LIB library is added for linking the programs.

To create the OS2NP.LIB library, we perform the following steps:

- Identify the unsupported APIs used by the applications.
- Prototype the API's functions in a header file.
- Code the functions using OS/2 equivalent functions.
- Put the functions into an object library.
- Link the applications with this library.

```
proj = server
cflags = /C /DOS2 /Gm+ /I..\os2np /N3 /Ss /Ti /Wgen /Wpro
lflags = /CO /O:$(proj) /PM:PM
objlist = $(proj).obj main.obj

.c.obj:
  icc $(cflags) $*.c

$(proj).exe : $(objlist) $*.res
  ilink $(lflags) $(objlist) pmwinx.lib ..\os2np\npipe.lib
  rc $*.res $*.exe

$(proj).res: $*.rc
  rc /r /dOS2 $*.rc
```

Figure 164. Named Pipe Server: MAKEFILE

```
proj = client
cflags = /C /DOS2 /Gm+ /I..\os2np /N3 /Ss /Ti /Wgen /Wpro
lflags = /CO /O:$(proj) /PM:PM
objlist = $(proj).obj main.obj

.c.obj:
  icc $(cflags) $*.c

$(proj).exe : $(objlist) $*.res
  ilink $(lflags) $(objlist) pmwinx.lib ..\os2np\npipe.lib
  rc $*.res $*.exe

$(proj).res: $*.rc $*.h
  rc /r /dOS2 $*.rc
```

Figure 165. Named Pipe Client: MAKEFILE

6.5.1 Unsupported API Function Classification

The unsupported named pipe API functions can be classified into two categories.

The functions in the first category are not defined in Open32. They are:

- CreateNamedPipe()
- ConnectNamedPipe()
- DisconnectNamedPipe()

Functions in the second category are defined in Open32 but do not support named pipes. They are:

- CreateFile()

- ReadFile()
- WriteFile()
- CloseHandle()

In the first case, we need to provide the function prototype and code while in the second case, we have to find a solution to enhance the existing functions to support the named pipe.

The technique we have chosen to handle both of these cases is to rename these functions in the NPIPE.H prototype file and to code the renamed functions in the NPIPE.C file using OS/2 equivalent API functions. One restriction imposed by this technique is that you cannot call these functions to access normal files and named pipes from the same application source code file. Instead you should separate them into different source code files and only include the NPIPE.H header file in the source code files which contain function calls for named pipe access. In our sample we are not calling any of these Open32 functions to perform tasks other than working with the named pipe, so this implementation will work.

6.5.2 Unsupported API Function Prototyping

The function prototyping task consists of:

- Defining the prototypes of the functions of the first category

  The prototypes of these functions can be found in the Win32 API function header files. You will want to model the function you are going to implement after these header file definitions and place them into the NPIPE.H header file.

- Renaming the functions of the second category

  The new name of these functions must not exist on Win32 nor on OS/2. Their prototypes must use the data and function types defined in Open32 since they will be called by Open32 applications. For the sample program presented in this redbook we did not rename these functions since the application only works with these functions for requests for named pipe activities. Thus, the function names could be used and all requests redirected to the functions provided by the API extensions implemented in the interface routine.

- Defining the data types referenced by the function prototypes

  Some of the data types may already exist in Open32. Thus you will need to define those data types that are absent in Open32.

- Defining the macros referenced by the function prototypes

Some of the macros may already exist in Open32. As with the data types you will only need to define those macros that are absent in Open32.

6.5.3 Unsupported API Function Coding

The implementation steps are the same for all the functions:

- Translate the input argument values of the Win32 API's functions into the corresponding OS/2 functions.
- Call the OS/2 equivalent function.
- Translate the return value of the OS/2 API's functions into the corresponding Win32 functions.
- Translate the output argument values of the OS/2 API's functions into the corresponding Win32 functions.
- If an error occurs, translate the OS/2 error code into a Win32 error code and store it to the thread making the call by calling SetLastError().

The NPIPE.C source code file contains only native OS/2 code. Because of the similar names between Open32 and native OS/2 function calls it cannot include the header file for Open32 (OS2WIN.H). Consequently, the definition of the macros and data types referenced by the Win32 API functions being implemented in the interface routine need to be duplicated at the beginning of the NPIPE.C source code file. Similarly, the SetLastError() function (a Win32 function whose prototype is defined in the os2win.h header file) calls the NPIPE.C source code file, and needs to be prototyped in the source file.

After the coding is finished, we put the NPIPE.OBJ object file into the NPIPE.LIB static function library. It will be included with the server and client sample programs when the OS/2 version of the application is link-edited.

```
proj = npipe
cflags = /C /Gm+ /N3 /Ss /Ti /Wgen /Wpro

 .c.obj:
  icc $(cflags) $*.c

$(proj).lib : $*.obj
  ilib $*.lib -+$*.obj;
```

Figure 166. Named Pipe Library MAKEFILE

6.6 Run the Applications

You run the named pipe server and client applications on a local machine or on a LAN. If running the application on a local machine it must be an OS/2 or Windows NT machine since Windows 95 does not support Win32 named pipe APIs. When running the application in a LAN environment you can run the named pipe server application on Windows NT or OS/2. The named pipe client applications can be run on Windows 95, Windows NT and OS/2. When running the application on a LAN, both the client and server machines must have installed networking support for remote named pipe support, for example OS/2 LAN Server for server machines and OS/2 LAN Requester for client machines.

Following are the steps to run the named pipe server and client applications:

1. Start the named pipe server application.
2. Go to the File pull-down menu and select the **Open** option to create a named pipe instance. Now the named pipe server application is waiting for the connection from a named pipe client.
3. Start the named pipe client application with the name of the server where the named pipe server application is running. If you do not give the server name, the named pipe is assumed to be local.
4. Go to the File pull-down menu and select the **Open** option to open the named pipe instance.
5. Once the connection is established between the named pipe server and client, you can use the **Send** and **Receive** options of the File pull-down menu, on both the server and client side, to exchange messages between them.
6. Repeat steps 2, 3 and 4 for each additional named pipe connection you wish to establish.
7. When you are finished exchanging messages use the **Close** option of the File pull-down on both the server and client, to close the connection.

Chapter 7. Tree View Control Sample Program

With Windows 95, Microsoft introduced a new set of controls to enhance the appearance of the user interface. These new window classes, called common controls, include:

- Toolbars
- Status Windows
- Property Sheets
- Tab Controls
- List Views
- Tree Views
- Rich Edit Controls
- Trackbars
- Progress Controls
- Header Controls
- Up-Down Controls
- Animation Controls
- Hot-Key Controls

While these new window classes give the Windows programmer greater power and flexibility in designing a user interface, they make applications more difficult to migrate to OS/2. Specifically, none of the above window classes are supported by Developer API Extensions, so applications which use them require extra work to migrate.

In this chapter, we will investigate an alternative to mixed mode programming to enable these common controls under OS/2 (See Chapter 5, “Mixed Mode Sample Program” on page 133 for more information on mixed mode programming). We have chosen to call the technique discussed in this chapter a translation control. It is intended primarily for migrating existing Win32 applications to OS/2 because it does not require any significant changes to the program source code (one line of code must be changed.) Our goal in developing this technique was to minimize the amount of source code required to migrate a Win32 application to OS/2. While the technique could also be used in developing new cross-platform applications, mixed mode programming may be a better solution.

To demonstrate the translation control technique, we have selected the Win32 tree view common control to implement.

In the first part of the chapter, we will discuss the translation technique and use it to implement the tree view control under OS/2. In the second part, we will use the tree view control to migrate TVTest, a program which

demonstrates most of the features of the tree view control, to OS/2. TVTest is show in Figure 167 on page 196 running under Windows 95. In the final part of the chapter, we will explain how to implement your own Open32 common controls using the translation technique.

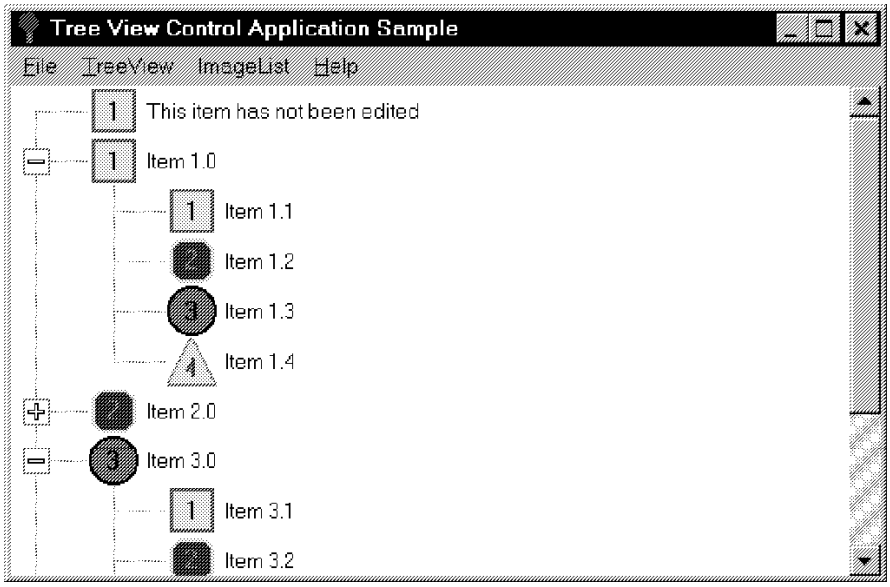


Figure 167. Tree View Control under Windows 95

7.1 How the OS/2 Tree View Control Works

The way a Win32 program interacts with the tree view control is visualized in Figure 168 on page 197. The application sends messages directly to the control, and the control sends notification messages directly to the application.

The approach used in this chapter to allow an Open32 application to use a common control under OS/2 is visualized in Figure 169 on page 197.

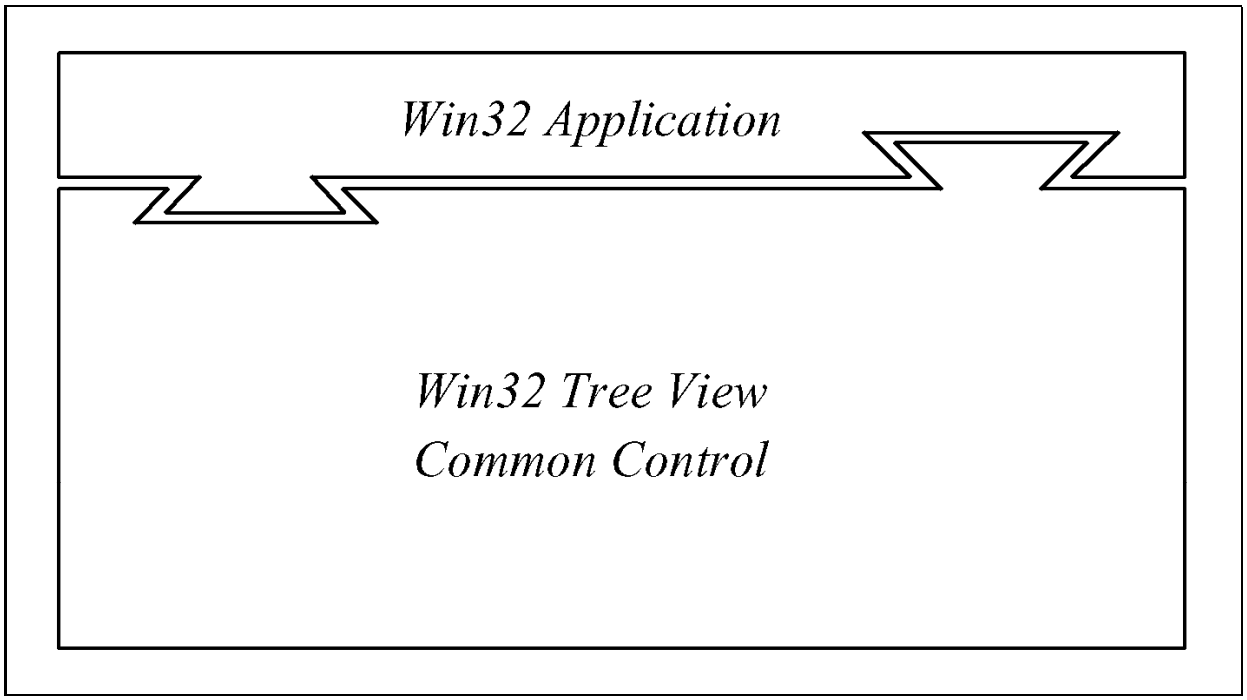


Figure 168. Tree View Control under Windows

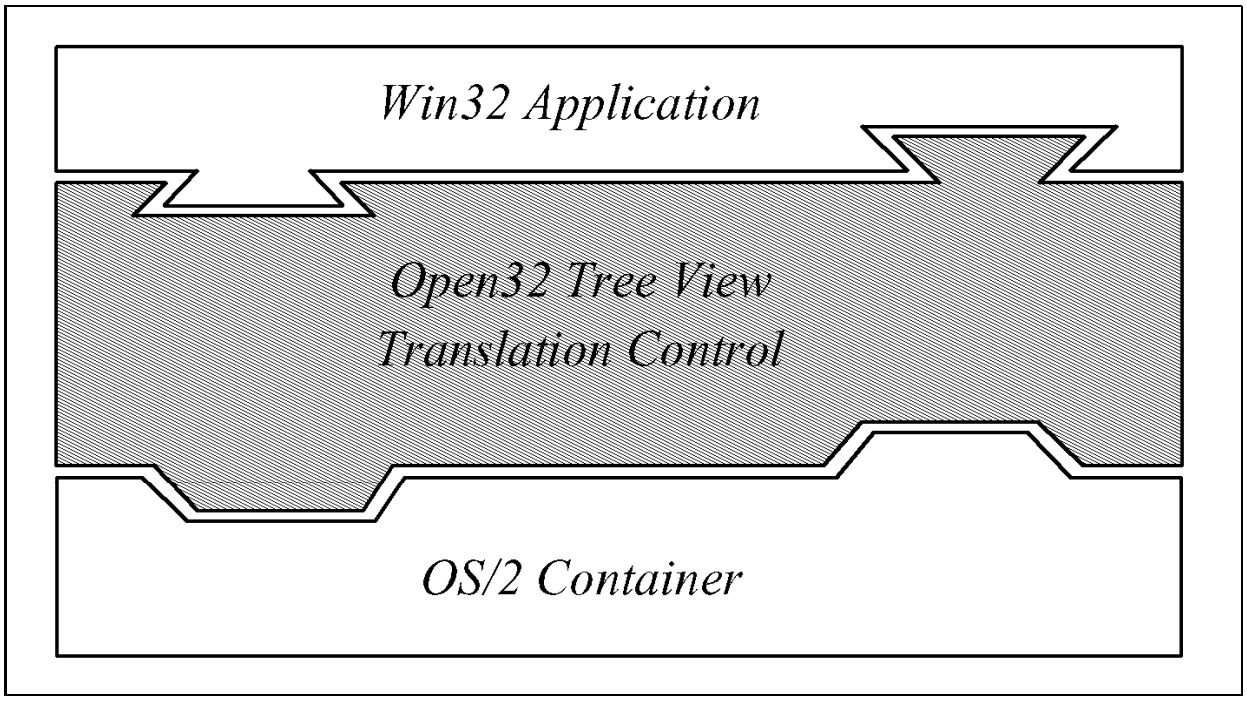


Figure 169. Tree View Translation Control under OS/2

The Win32 application interacts with the Open32 tree view control just as it interacts with the tree view under Windows. The Open32 tree view control, however, acts as a translator between the application and an OS/2 container control. The container is a standard OS/2 window class which provides, among other things, a tree view of data. Every message the Win32 application sends to the Open32 tree view is translated into an equivalent container message and sent to the container. Notification messages from the container are translated and sent to the Win32 application.

7.1.1 Translation Technique Advantages

The translation technique has the following advantages over mixed mode programming:

- The original program code does not change because all of the code for handling the differences between the tree view and container is isolated in the translation control. This makes the technique an ideal solution for existing programs which use common controls. By adding translation controls for any common controls the application uses, the original program source can be left unchanged.
- The Open32 translation control can be packaged as a DLL and shared by several running applications. This reduces memory requirements if more than one application uses the control.
- Experienced Windows programmers do not need to learn OS/2 programming to use the control. (A programmer familiar with both Windows and OS/2 is needed, however, to write the control.)
- Once written the first time, the control can be reused in the future without modification. It can be copied from one project to another or can be sold to other companies and developers.

7.1.2 How the OS/2 Tree View Control is Written

The key to making the translation control work is writing an effective translation window class. This custom window class will allow the Win32 application to use an OS/2 control through the standard Win32 common controls interface.

To build a translation window class, the programmer must deal with Win32 and OS/2 APIs, messages, and data structures. The translation window class must call both OS/2 and Win32 functions. Thus, the translation window class source code must include both <os2.h> and <os2win.h>.

The need to include both header files causes a problem. The header files os2.h and os2win.h both define structures and messages with the same name (but different meanings or values). Having these duplicate definitions

in the source code will cause compile-time errors. A translation window class eliminates the compiler errors by separating the source code in two source code files. One file, called the Open32 source code, will be compiled with the <os2win.h> header file. The other file, called the OS/2 source code, will be compiled with the <os2.h> header file.

However, there are times when the OS/2 code needs to call an Open32 function and vice versa. For these instances, you must create a function which calls the Open32 function for the OS/2 code. An example of this in the Open32 tree view control is Open32SendMsg(), which calls the Win32 function SendMessage().

7.1.2.1 Open32 Source Code

The Open32 source code for the tree view control is contained in the file OPEN32TV.C, which can be found on the CD in the directory TVTEST OPEN32TV. The file contains four procedures, as described in Table 7.

Table 7. Procedures Defined in OPEN32TV.C

Function	Purpose
void InitTreeView(HINSTANCE hInst)	Registers the WC_TREEVIEW window class using the Open32 API RegisterClass()
LRESULT CALLBACK TVOpen32WndProc(HWND hwnd, UINT message, WPARAM wParam, LPARAM lParam)	Open32 window procedure for the WC_TREEVIEW window class
LONG Open32SendMsg(HWND hwnd, ULONG msg, ULONG wp, ULONG lp)	Sends a message using the Open32 API SendMessage()
ULONG Open32GetLong(HWND hwnd, ULONG index)	Retrieves a window long using the Open32 API GetWindowLong()

The first function, InitTreeView(), is similar in purpose to the Win32 function InitCommonControls(). While it would make sense to name the function InitCommonControls(), the Win32 function takes no parameters, whereas InitTreeView() requires an instance handle as a parameter. The instance handle must be passed to RegisterClass(). To reduce confusion, another name has been selected. Note, however, that either decision will mandate a small change to the Win32 program's source code. See 7.2.2, “Changes to the Source Code” on page 214 for more information on this change. This is the only change which is required for all window classes which are implemented using the method outlined in this chapter, although the tree view example will also require a few other small changes.

The second function, TVOpen32WndProc(), is an Open32 window procedure which handles the messages sent to the translation window by the Open32 application. It calls MsgToTV(), a function defined in OS2TV.C, which performs message translation. No message translation occurs in TVOpen32WndProc.

The third and fourth functions directly call the Open32 API they are associated with. Open32SendMsg() calls SendMessage, and Open32GetLong() calls GetWindowLong(). These two functions are called by functions in OS2TV.C.

7.1.2.2 OS/2 Source Code

The OS/2 source code for the tree view control is contained in the file OS2TV.C, which can be found on the CD in \TVTEST\OPEN32TV. The file defines five procedures, as described in Table 8.

Table 8. Procedures Defined in OS2TV.C

Procedure	Purpose
LONG MsgToTV(HWND hwnd, ULONG msg, LONG wp, LONG lp, PTVWND DATA pTVWndData)	Processes messages from the parent window by translating them and sending equivalent container control messages.
MRESULT EXPENTRY TVOS2WndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)	OS/2 window procedure which is registered in MsgToTV during the processing of WM_CREATE. This procedure receives messages from the container control. They are translated and sent to the Open32 parent window.
void UpdateTreeImages (HWND hwnd, PRECORDCORE rec, PTVWNDDATA pTVWndData, PCNRINFO pCnrInfo)	Utility function called by MsgToTV(). This function updates all the images in the container after a new image list has been selected.
void UpdateImages (PRECORDCORE rec, PTVWNDDATA pTVWndData, PCNRINFO pCnrInfo)	Utility function called by MsgToTV() and UpdateTreeImages(). This function updates the image data for one item in the container.
ULONG CountRecords (HWND hwnd, PRECORDCORE rec, PCNRINFO pCnrInfo)	Recursively counts the records in the container. This function is called by MsgToTV().

The MsgToTV() and TVOS2WndProc() functions contain essentially all of the translation code. MsgToTV() receives all of the messages sent by the Win32 application to the Open32 tree view control. The function processes each message individually, translating it into its OS/2 container equivalent. See 7.1.4.1, “TVM_INSERTITEM Message” on page 205 for more information on how MsgToTV() translates messages.

TVOS2WndProc() receives messages directly from the container. It translates OS/2 WM_CONTROL messages into equivalent Win32 WM_NOTIFY messages. See 7.1.4.2, “CN_CONTEXTMENU Message” on page 208 for more information on how the TVOS2WndProc() receives messages from the container.

The utility functions UpdateTreeImage(), UpdateImages(), and CountRecords() perform various actions on the container control. UpdateTreeImage() and UpdateImage() provide code to simplify the handling of image lists, which are not used by the container control. CountRecords() counts all of the records in the container. Although the OS/2 container has a message to return the number of items currently in it, the container only returns the number of root level items. The Win32 treeview control API returns the total number of items in the item count message. This action is duplicated by CountRecords().

7.1.3 Overview of the Translation Process

As stated before, the translation process is the key to the entire method. Two functions, both defined in OS2TV.C, contain essentially all of the translation code. MsgToTV() manages the messages sent to the translation control by the Win32 program, and TVOS2WndProc() manages messages sent to the translation control by the container control.

For this discussion, we will differentiate between two different kinds of messages that are processed by the translation control.

- Command messages are sent by the Win32 parent window to the translation control to either retrieve information on one of the tree items or to request some action of the translation control. These messages are translated into OS/2 container control messages and sent to the translation control's container child window.
- Notification messages are sent by the container to the translation control to inform it of a user's actions. These messages are translated into Win32 notification messages and sent to the Win32 parent window.

Before going into the details of how the tree view translation control processes messages, let's look at the general flow of a command message and a notification message processed by the tree view translation control. Detailed information on how these messages are handled will be covered in 7.1.4, “Details on How the Tree View Control Works” on page 205.

7.1.3.1 Command Message Flow Example

To show the flow of a command message through the tree view translation control, we will use the Win32 TVM_INSERTITEM command message. The TVM_INSERTITEM message instructs the tree view to add a new item to its

contents. An outline of the flow of this message through the tree view translation control is shown in Figure 170 on page 202.

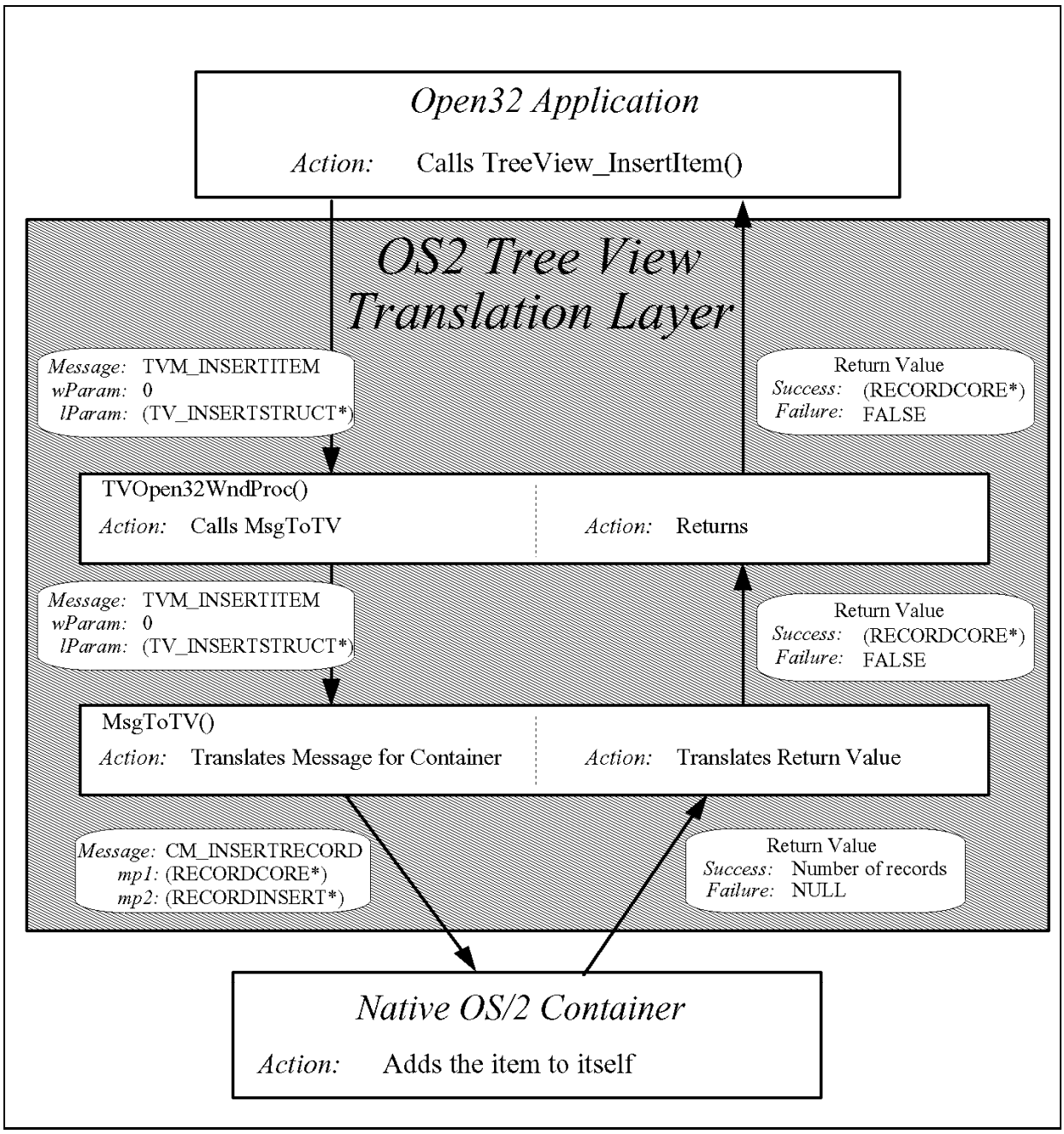


Figure 170. Command Messages Translated for the Tree View Control

The message begins when the Open32 application calls TreeView_InsertItem(). This macro sends the tree view a TVM_INSERTITEM

message, passing as the lParam a pointer to a TV_INSERTSTRUCT. The translation control receives the message and creates equivalent OS/2 RECORDCORE and RECORDINSERT structures. It then sends the OS/2 container control an CM_INSERTRECORD message.

The container processes the CM_INSERTRECORD message and returns the number of records in the container. If the return is not zero, the record was properly inserted and the translation window will return a handle to the inserted item. If the return is zero, an error occurred and the translation window will return FALSE.

Notice that the return types of the container and the Win32 TreeView APIs are different. Both return FALSE or zero when an error occurs, but they return different positive values to signal success. The translation control must handle these differences and convert them appropriately. The specifics of this conversion are covered in 7.1.4, “Details on How the Tree View Control Works” on page 205.

7.1.3.2 Notification Message Flow Example

When the container sends a notification message, the tree view translation control must also translate it into an equivalent tree view message to send to the parent window. To show the flow of a notification message through the tree view translation control, we will use the OS/2 control WM_CONTROL message CN_CONTEXTMENU. See Figure 171 on page 204 for a graphical representation of how the Open32 tree view control does this.

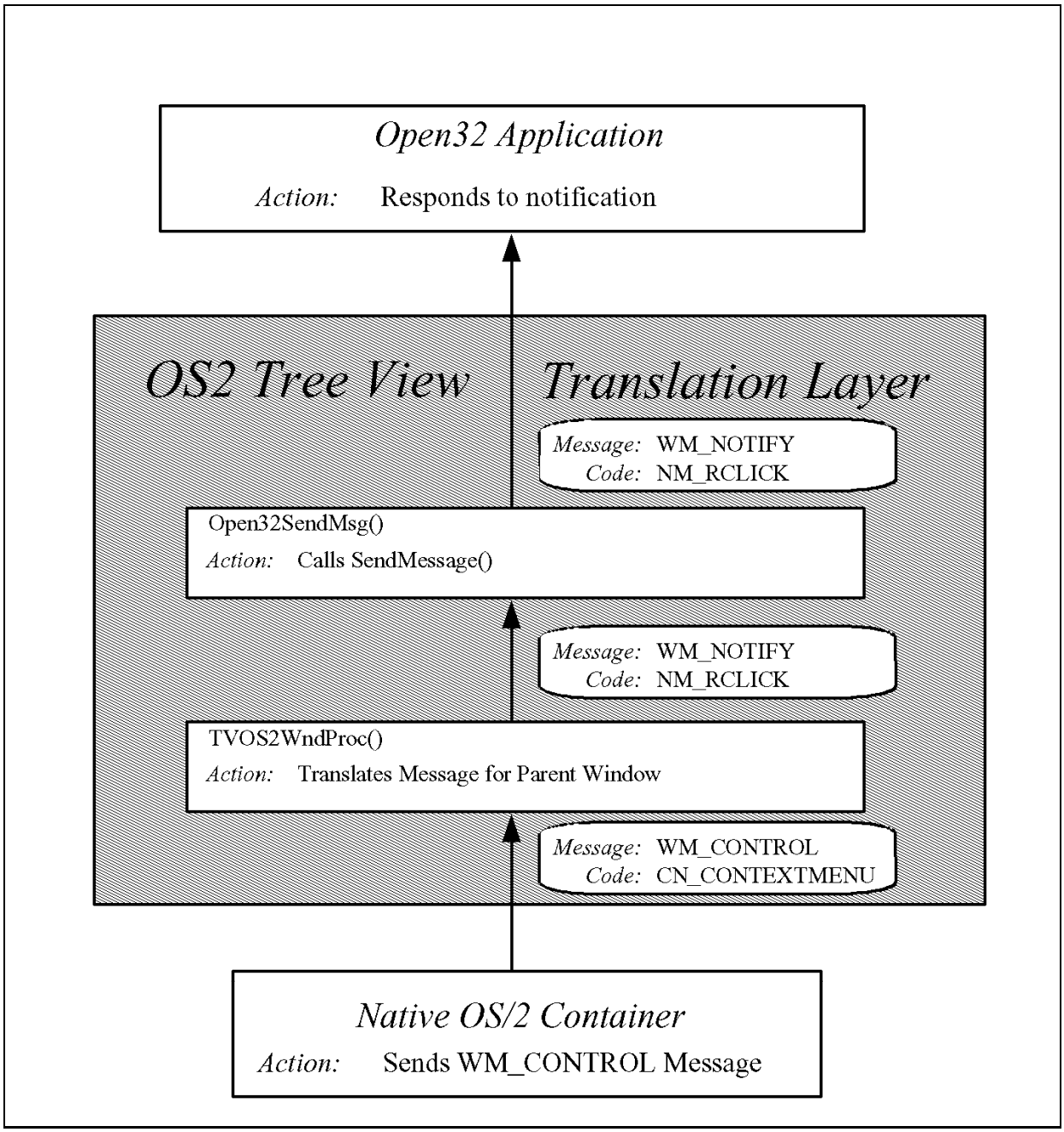


Figure 171. Notification Messages Translated by the Open32 Tree View Control

When the container sends the translation control a WM_CONTROL message, such as CN_CONTEXTMENU, OS/2 calls TVOS2WndProc() to process the message. The window procedure translates the CN_CONTEXTMENU into a Win32 NM_RCLICK notification message. It then sends the NM_RCLICK

message as a WM_NOTIFY message to the Win32 application, which responds appropriately.

7.1.4 Details on How the Tree View Control Works

In this section we will take a close look at how the tree view translation control works. First, we will step through the code which handles the TVM_INSERTITEM message from an Win32 application. Then we will see exactly how it translates the CN_CONTEXTMENU message from the container.

7.1.4.1 TVM_INSERTITEM Message

The message begins when the Win32 application creates and initializes a TV_INSERTSTRUCT for the item to be added. It then calls TreeView_InsertItem() to add the item to the tree view. This macro is not supported by Open32, so we must provide a definition of it, as shown in Figure 172. The macro is defined in OS2WINTV.H, which is on the CD in \TVTEST\OPEN32TV.

```
#define TreeView_InsertItem(hwnd, lpis) \
   (HTREEITEM)SendMessage((hwnd), TVM_INSERTITEM, 0, (LPARAM)(LPTV_INSERTSTRUCT)(lpis)
```

Figure 172. OS2WINTV.H: Definition of TreeView_InsertItem()

The macro calls the Win32 function SendMessage(), which is supported by Open32, passing zero as the wParam and a pointer to the TV_INSERTSTRUCT as the lParam. The Developer API Extensions of OS/2 Warp calls the tree view class's window procedure TVOpen32WndProc() to handle the message, passing the parameters to it as they are given to SendMessage(). Since the parameters are passed unchanged, no translation occurs in this step. Note that Developer API Extensions of OS/2 Warp does *not* call TVOS2WndProc(), which as we will see is the subclassed window procedure for the translation window class. Instead, Developer API Extensions calls TVOpen32WndProc(), which is the window procedure which was originally specified when the class was registered with it by the call to RegisterClass().

```
LRESULT CALLBACK TVOpen32WndProc(HWND hwnd, UINT message,
                     WPARAM wParam, LPARAM lParam)
{
   PTVWNDDATA pTVWndData;
   if (message != WM_CREATE) {
      pTVWndData = (PTVWNDDATA) GetWindowLong(hwnd, 0);
   }
   switch(message)
   {
   . . .
      case TVM_INSERTITEM:
      {
         return MsgToTV(hwnd, TVM_INSERTITEM, wParam, lParam, pTVWndData);
         break;
      }
   . . .
   }
}
```

Figure 173. OS2TV.C: Processing the TVM_INSERTITEM Message

The code for TVOpen32WndProc() is shown in Figure 173. The window procedure retrieves the pointer to the window's data using the Win32 API GetWindowLong(). The window data contains window-specific information, such as its window ID, the window handle of its container, and the handle to its image list. Data is stored in the window long which allows the code to be reentrant. TVOpen32WndProc() then calls MsgToTV(), passing the window handle, message, message parameters, and pointer to the window data.

```
 1: LONG MsgToTV(HWND hwnd, ULONG msg, LONG wp, LONG lp, PTVWNDDATA pTVWndData)
 2: {
 3:    switch(msg)
 4:    {
 5:       . . .
 6:       case TVM_INSERTITEM:
 7:       {
 8:          TV_INSERTSTRUCT *ins;
 9:          PRECORDCORE rec;
10:          RECORDINSERT recsIn;
11:          CNRINFO CnrInfo;
12:          ULONG x;
13:          ins = (TV_INSERTSTRUCT *) PVOIDFROMMP(lp);
```

Figure 174 (Part 1 of 2). OS2TV2.C: Processing the TVM_INSERTITEM Message

```
14:         CnrInfo.cb = sizeof(CnrInfo);
15:         WinSendMsg(pTVWndData->hwndContainer, CM_QUERYCNRINFO,
16:                  MPFROMP(&CnrInfo), MPFROMLONG(sizeof(CnrInfo)));
17:
18:         rec = (PRECORDCORE) WinSendMsg(pTVWndData->hwndContainer, CM_ALLOCRECORD,
19:                                  0, MPFROMLONG(1L));
20:         rec->flRecordAttr = 0l;
21:         rec->ptlIcon.x =0;
22:         rec->ptlIcon.y =0;
23:         rec->preccNextRecord = NULL;
24:         rec->pszIcon = 0L;
25:         rec->pTreeItemDesc = (TREEITEMDESC*) malloc(sizeof(TREEITEMDESC));
26:         rec->hptrMiniIcon = ins->item.iImage;
27:         rec->hbmMiniBitmap = ins->item.iSelectedImage;
28:         UpdateImages(rec, pTVWndData, &CnrInfo);
29:         rec->pszText = (PSZ) ins->item.lParam;
30:         rec->pszName = NULL;
31:         rec->pszTree = strdup(ins->item.pszText);
32:         recsIn.cb = sizeof(recsIn);
33:         if (ins->hParent == TVI_ROOT)
34:           recsIn.pRecordParent = 0;
35:         else
36:           recsIn.pRecordParent = ins->hParent;
37:         switch((ULONG)ins->hInsertAfter)
38:         {
39:            case (ULONG)TVI_FIRST:
40:              recsIn.pRecordOrder=(PRECORDCORE)CMA_FIRST;
41:              break;
42:            case (ULONG)TVI_LAST:
43:              recsIn.pRecordOrder=(PRECORDCORE)CMA_END;
44:              break;
45:            default:
46:              recsIn.pRecordOrder=ins->hInsertAfter;
47:         }
48:         recsIn.fInvalidateRecord=TRUE;
49:         recsIn.zOrder=CMA_TOP;
50:         recsIn.cRecordsInsert=1;
51:         if(!WinSendMsg(pTVWndData->hwndContainer, CM_INSERTRECORD,
52:                      MPFROMP(rec), MPFROMP(&recsIn)))
53:         {
54:            //Error!!!
55:            return FALSE;
56:         } else {
57:            //Ok!
58:            return (LONG) rec;
59:         }
60:       }
61:       . . .
62:    }
63: }
```

Figure 174 (Part 2 of 2). OS2TV2.C: Processing the TVM_INSERTITEM Message

The source code for MsgToTV() is shown in Figure 174 on page 206. Line numbers have been added for reference purposes. The function creates a new RECORDCORE by sending the container a CM_ALLOCRECORD message to allocate the memory (line 18). MsgToTV() then copies

information from the TV_INSERTSTRUCT into the RECORDCORE, translating specifications such as the item's parent (lines 33-36) and previous sibling (lines 37-47). MsgToTV() also creates a RECORDINSERT structure and fills in the values for it.

Several pieces of information exist in the TV_ITEM structure (a part of the TV_INSERTSTRUCT) which do not exist in the RECORDCORE. To store this information, the tree view translation control takes advantage of the fact that the container supports many features which are never exercised by the tree view. First, the tree view must store the indices of the item's image and selected image. It makes use of two unused RECORDCORE members, *hptrMiniIcon* and *hbmMiniBitmap* to do so (lines 26-27). The mini images are never used by the container in tree mode, so there is essentially no risk of system or data corruption. Second, 32-bit value *lParam* is available for Win32 applications to use for private data storage. The value is stored in *pszText* (line 29), which is normally used to store the address of a character string. The string is only used in the container's text view, so again it is basically safe to misuse the data member.

After the RECORDCORE and RECOREINSERT structures are created, the function sends a CM_INSERTRECORD message to the container (lines 51-52). If the container returns 0, an error occurred and the function returns FALSE (lines 54-55). Otherwise, the item was added correctly and the function returns the address of the RECORDCORE (lines 57-58).

The RECORDCORE address serves as the handle to the tree view item (HTREEITEM). Because the logical structure of HTREEITEM is conveniently not defined in Win32, applications cannot modify items directly. Therefore, the actual value and meaning of HTREEITEM is not important for the Win32 application. HTREEITEM is defined in OS2WINTV.H as a pointer to a RECORDCORE, as shown in Figure 175.

```
#ifdef _OS2WIN_H
typedef struct _RECORDCORE RECORDCORE, *PRECORDCORE;
#endif

typedef PRECORDCORE HTREEITEM;
```

Figure 175. OS2WINTV.H: Definition of HTREEITEM

7.1.4.2 CN_CONTEXTMENU Message

When the user clicks the second mouse button on a container, the control sends its parent a CN_CONTEXTMENU message so it can display a context-sensitive pop-up menu. In this section, we will see how the Open32

tree view control receives and translates this message for its parent window.

The message begins when the container sends a WM_CONTROL message with the notifycode set to CN_CONTEXTMENU. OS/2 calls the tree view's window procedure, TVOS2WndProc(), to process the message. Note that the container's message is *not* sent to TVOpen32WndProc().

Figure 176 shows the section of TVOS2WndProc() which processes the CN_CONTEXTMENU message.

```
MRESULT EXPENTRY TVOS2WndProc(HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)
{
   PTVWNDDATA pTVWndData;
   pTVWndData = (PTVWNDDATA) Open32GetLong(hwnd, 0);
   switch (msg)
   {
      case WM_CONTROL:
      {
         switch(SHORT2FROMMP(mp1))
         {
             . . .
             case CN_CONTEXTMENU:  // Sent when user presses mouse button
                                   // 2 to open a context-sensitive menu
             {
                NMHDR nh;
                nh.hwndFrom = hwnd;
                nh.idFrom = pTVWndData->WindowID;
                nh.code = NM_RCLICK;
                Open32SendMsg(pTVWndData->hwndWinParent, 0x004E, // WM_NOTIFY
                   pTVWndData->WindowID, (ULONG) MPFROMP(&nh));
                break;
             }
             . . .
         }
      }
      default:
      {
         return pTVWndData->oldWP(hwnd, msg, mp1, mp2);
      }
   }
   return FALSE;
}
```

Figure 176. OS2TV.C: TVOS2WndProc() Processing CN_CONTEXTMENU

The tree view control must first retrieve the information about itself. The data is stored in a PTVWNDDATA structure, and a pointer to each tree view control's data structure is stored in its window longs. Because the window long is stored by Open32, the tree view control must call Open32GetLong() to retrieve the pointer; calling the OS/2 API WinQueryWindowULong() will not work because OS/2 APIs cannot access the window long stored by

Open32. Open32GetLong() is defined in OPEN32TV.C, as shown in in Figure 177 on page 210.

```
ULONG Open32GetLong(HWND hwnd, ULONG index)
{
   return (ULONG) GetWindowLong(hwnd, index);
}
```

Figure 177. OPEN32TV.C: Open32GetLong()

TVOS2WndProc() then creates a NMHDR (notify message header), which is a Win32 structure. The field *nh.code* is set to NM_RCLICK, which means that the user has pressed the right mouse button. Note that a slight assumption is at work here. The tree view cannot know what the application will do with the NM_RCLICK message, but it is a reasonable assumption that it may display a context menu, since most Win32 programs use the second mouse button to display context menus.

After the NMHDR has been created, the tree view sends a WM_NOTIFY message to its parent window. Note that it cannot use WinSendMsg() to do this, as the message will be filtered and removed by the Open32 translation layer. Therefore, it calls Open32SendMsg() to send the message.

Open32SendMsg() is defined in OPEN32TV.C, as shown in Figure 178.

```
ULONG Open32SendMsg(HWND hwnd, ULONG msg, ULONG wp, ULONG lp)
{
   return SendMessage(hwnd, msg, wp, lp);
}
```

Figure 178. OPEN32TV.C: Open32SendMsg()

The function simply calls the Open32 function SendMessage(). It does not translate the message. Since SendMessage() is an Open32 function, it will directly call the parent window's window procedure without filtering the message. In this way, the tree view control can send the WM_NOTIFY message to its parent window.

7.1.5 Handling the Image List

In Win32 programming, an image list is required by tree view to store the bitmaps or icons associated with tree view items. The Win32 application is responsible for creating and populating the image list. It sends a message to tree view to set the image list.

While this can help reduce resource usage in Windows (the image list is designed to be resource efficient), it complicates the creation of the tree

view translation control. The container control does not use anything like the image list. To maintain source-level compatibility with Win32 applications, the tree view translation control must also support an image list.

For this reason, a basic implementation of an image list was created to accompany the tree view translation control. The OS/2 image list differs slightly from the Win32 image list, and thus it requires a few small changes to the Win32 source code.

The source code for the OS/2 image list is in the file OPEN32IL.C in the same directory as the tree view translation control source code. It also has a header file, OS2WINIL.H.

OPEN32IL.C contains several functions, as outlined in Table 9.

Table 9 (Page 1 of 2). Functions Defined in OPEN32IL.C

Function	**Purpose**
HIMAGELIST **ImageList_Create** (int cx, int cy, UINT flags, int cInitial, int cGrow)	Creates a new image list with *cInitial* images of width *cx* and height *cy*. The *flags* and *cGrow* parameters are stored internally for future reference.
void **ImageList_Grow**(HIMAGELIST himl)	Increases the size of the image list by the amount specified when the image list was created. The image list will automatically call this function when necessary; applications and translation controls should never call it directly.
int **ImageList_Add** (HIMAGELIST himl, ULONG idImage, ULONG idMask)	Adds a bitmap to the image list. The *idImage* parameter specifies the numeric ID of the bitmap in the executable file; *idMask* specifies the bitmap mask to be used with the image.
int **ImageList_Replace** (HIMAGELIST himl, int i, ULONG idImage, ULONG idMask)	Replaces a bitmap in the image list. The *i* parameter specifies the index of the image to be replaced; *idImage* specifies the numeric ID of the new bitmap in the executable file; *idMask* specifies the new bitmap mask to be used with the image.
int **ImageList_AddIcon** (HIMAGELIST himl, ULONG idImage)	Adds an icon to the image list. The *idImage* parameter specifies the numeric ID of the icon in the executable file.
int **ImageList_ReplaceIcon** (HIMAGELIST himl, int i, ULONG idImage)	Adds an icon to the image list. The *i* parameter specifies the image to replace and *idImage* specifies the numeric ID of the new icon in the executable file.

Table 9 (Page 2 of 2). Functions Defined in OPEN32IL.C

Function	Purpose
ULONG **ImageList_GetImage** (HIMAGELIST himl, int index)	Retrieves the handle to the image *index*. This function is used by the Open32 tree view control and has no counterpart in the Win32 image list.
ULONG **ImageList_GetMask** (HIMAGELIST himl, int index)	Retrieves the handle to the image mask *index*. This function is used by the Open32 tree view control and has no counterpart in the Win32 image list.
IMAGETYPE **ImageList_GetType** (HIMAGELIST himl)	Returns the type of images in the image list. Valid values are *icon*, *bitmap*, and *unknown*. This function is used by the Open32 tree view control and has no counterpart in the Win32 image list.
BOOL **ImageList_GetSize** (HIMAGELIST himl, short * cx, short * cy)	The height and width of the images in the image list are put in *cx* and *cy*, respectively. This is equivalent to ImageList_GetIconSize(), except that short integers are used. This function is used by the Open32 tree view control.
BOOL **ImageList_Remove**(HIMAGELIST himl, int index)	The image *index* is removed from the list and its resources are freed from memory.
ULONG **ImageList_GetStyle**(HIMAGELIST himl)	Returns the style flag that was specified when the image list was created. This can be used by translation controls to determine if the image list uses masks.
BOOL **ImageList_Destroy** (HIMAGELIST himl)	Destroys the image list, freeing all memory and resources.

Although the Open32 image list supplies only the basic image list functions, it may serve many applications very well. It allows either bitmaps or icons of any size to be inserted into the image list, although the two cannot both be present in one image list. It will automatically grow as needed.

The Win32 image list works by copying images added to it into a single large bitmap. On the other hand, the OS/2 image list works by simply calling WinLoadIcon or GpiLoadBitmap to load the bitmap or icon from the application's executable file. This assumes that the resources are not bound to another file or created at run-time. While loading the resources individually in OS/2 will use many image handles, OS/2 does not restrict the number of open images which Windows 95 does. Therefore, the only limitation on the number of images that an image list can contain is the availability of memory.

7.1.5.1 OS/2 Image List Precautions

Because the focus of this chapter is on the tree view translation control, the image list contains only the functions needed for the tree view control to work. While it provides functions for adding and replacing images and will dynamically grow, it performs very little error checking to ensure function parameters are valid. An incorrect parameter passed to an image list function could easily generate a memory access violation and cause OS/2 to terminate the program.

The OS/2 image list may require modification before it will work for your application. One important thing to keep in mind when modifying the image list is that image handles returned by Open32 functions cannot be used with OS/2 functions. Therefore, handles returned by LoadIcon() and LoadBitmap() cannot be passed to the OS/2 container control or to any other OS/2 control. While doing so probably will not cause any system errors (we never experienced any during development), the image will not appear.

7.2 Using the Tree View Translation Control

In this section we will use the tree view translation control, described in 7.1, “How the OS/2 Tree View Control Works” on page 196, to migrate a sample application. The sample program, TVTest, demonstrates the tree view's capabilities and allows the user to experiment with the tree view's various functions.

The TVTest program was initially written for Windows 95 to demonstrate the tree view control. Although it was written expressly to be used as a migration sample, no special consideration was given at the time it was written to simplify the migration. By using the tree view translation control, very few changes are necessary to migrate it to OS/2.

7.2.1 Copying the Source Files

The Win32 source code for the TVTest program is on the CD in the directory TVTEST WIN32. We recommend that you copy the files from there to your hard drive in OPEN32 TVTEST MIGRATE so you can convert the original Win32 program into a new OS/2 program. You will also need the Open32 tree view control library, which is on the CD in TVTEST OPEN32TV and should be copied to OPEN32 TVTEST OPEN32TV. A completely migrated TVTest application is on the CD in TREEVIEW OS2.

7.2.2 Changes to the Source Code

In line with the goals for the Open32 tree view control, very few changes are required to the program source code.

7.2.2.1 Changing Header Files

The Win32 program must include different header files for Open32 compilation. This change is shown in Figure 179.

```
#ifndef OS2
#include <windows.h>
#include <commctrl.h>
#include "tvtest.h"
#else
#include <os2win.h>
#include <string.h>
#include <malloc.h>
#include <os2wintv.h>   // Open32 Tree View control
#include <os2il.h>     // Open32 Image List support
#include "tvtest.h"
#endif
```

Figure 179. TVTEST.C: Changes to the #include Statements

The way OS/2 header files correspond to Win32 header files is shown in Table 10.

Table 10. Header File Correspondence between Windows and OS/2

Windows Header File	OS/2 Header File	Explanation
<windows.h>	<os2win.h>	Open32 header file replaces the standard Windows header file.
none	<string.h> <malloc.h>	While Microsoft Visual C++ automatically includes several standard header files, VisualAge C++ does not. As a result, they must be manually specified for OS/2 compilation.
<commctrl.h>	<os2wintv.h> <os2il.h>	The <commctrl.h> header file defines all of the Win32 common controls, including the tree view and image list. The OS/2 tree view control is defined in <os2wintv.h> and a basic image list is defined in <os2il.h>.
"tvtest.h"	"tvtest.h"	The same file defines the numeric resource IDs.

7.2.2.2 Changing InitCommonControls()

The Win32 API InitCommonControls() must be changed to InitTreeView(). This change is shown in Figure 180.

```
#ifdef OS2
          InitTreeView(hInst);
#else // Windows
          InitCommonControls();
#endif
```

Figure 180. Changes to TVTEST.C

Note

All of the changes shown in this chapter are made using precompiler `#ifdefs`. While excessive precompiler statements can create confusing code, a few small `#ifdefs`, such as those in Figure 179 on page 214 and Figure 180, can allow small differences between platforms to be maintained in the same file. This simplifies program maintenance, since changes need to be made in only one file.

7.2.2.3 Changing ImageList Functions

The TVTest sample program uses an image list, as required by the Win32 tree view control. In this section we will discuss the changes necessary to migrate the image list. See 7.1.5, “Handling the Image List” on page 210 that covers the OS/2 image list and how it differs from the Win32 image list.

The sample image list for OS/2 requires the Win32 application add images by passing ImageList_Add() a resource ID, instead of an image handle as used by the Win32 image list. See 7.1.5, “Handling the Image List” on page 210 for more information on this difference. The Win32 application code for initializing the images in the image list must be changed to pass the resource ID instead of the image handle.

The original code for adding the icons is shown in Figure 181 on page 216.

```
case ID_IMAGELIST_ICONS:
   {
      HIMAGELIST old, Icons;
      HICON hIcon;
      Icons = ImageList_Create(32, 32, ILC_COLOR4 | ILC_MASK, 4, 4);
      if (!Icons)
         break;
      hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_RECTANGLE));
      ImageList_AddIcon(himl, hIcon);
      hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ROUND));
      ImageList_AddIcon(himl, hIcon);
      hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_CIRCLE));
      ImageList_AddIcon(himl, hIcon);
      hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_TRIANGLE));
      ImageList_AddIcon(himl, hIcon);
      old = TreeView_SetImageList(hwndTV, Icons, TVSIL_NORMAL);
      if (old)
         ImageList_Destroy(old);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_ICONS, MF_CHECKED);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_BITMAPS, MF_UNCHECKED);
      break;
   }
```

Figure 181. TVTEST.C: Original Code for Adding Images to Image List

In the original program, this code is executed during the processing of the ID_IMAGELIST_ICONS message. Since all the lines which load and add icons would need to be changed for OS/2, the decision was made to isolate the image list initialization in another function. The function AddIcons() is system dependent and is an example of traditional mixed mode programming.

Similar changes are needed for the handling of the ID_IMAGELIST_BITMAPS message. Another system-dependent function named AddBitmaps() loads the bitmap images.

The complete definition of AddImages() and AddBitmaps() is shown in Figure 182 on page 217. The revised code for processing ID_IMAGELIST_ICONS and ID_IMAGELIST_BITMAPS is shown in Figure 183 on page 219.

```
#ifdef OS2
BOOL AddIcons(HIMAGELIST himl)
{
   ImageList_AddIcon(himl, IDI_RECTANGLE);
   ImageList_AddIcon(himl, IDI_ROUND);
   ImageList_AddIcon(himl, IDI_CIRCLE);
   ImageList_AddIcon(himl, IDI_TRIANGLE);
   return TRUE;
}

BOOL AddBitmaps(HIMAGELIST himl)
{
    ImageList_Add(himl, IDB_BITMAPA, IDB_BITMAPMASK);
    ImageList_Add(himl, IDB_BITMAPB, IDB_BITMAPMASK);
    ImageList_Add(himl, IDB_BITMAPC, IDB_BITMAPMASK);
    ImageList_Add(himl, IDB_BITMAPD, IDB_BITMAPMASK);
    return TRUE;
}
#else  // ***************** WINDOWS *************
BOOL AddIcons(HIMAGELIST himl)
}
   HICON hIcon;
   hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_RECTANGLE));
   ImageList_AddIcon(himl, hIcon);
   DeleteObject(hIcon);
   hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_ROUND));
   ImageList_AddIcon(himl, hIcon);
   DeleteObject(hIcon);
   hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_CIRCLE));
   ImageList_AddIcon(himl, hIcon);
   DeleteObject(hIcon);
   hIcon = LoadIcon(hInst, MAKEINTRESOURCE(IDI_TRIANGLE));
   ImageList_AddIcon(himl, hIcon);
   DeleteObject(hIcon);
   return TRUE;
}
```

Figure 182 (Part 1 of 2). TVTEST.C: AddIcons() for both Windows and OS/2

```
BOOL AddBitmaps(HIMAGELIST himl)
{
   HBITMAP hbm, mask;
   mask = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAPMASK));
   hbm = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAPA));
   ImageList_Add(himl, hbm, mask);
   DeleteObject(hbm);
   hbm = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAPB));
   ImageList_Add(himl, hbm, mask);
   DeleteObject(hbm);
   hbm = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAPC));
   ImageList_Add(himl, hbm, mask);
   DeleteObject(hbm);
   hbm = LoadBitmap(hInst, MAKEINTRESOURCE(IDB_BITMAPD));
   ImageList_Add(himl, hbm, mask);
   DeleteObject(hbm);
   DeleteObject(mask);
   return TRUE;
}
#endif
```

Figure 182 (Part 2 of 2). TVTEST.C: AddIcons() for both Windows and OS/2

```
case ID_IMAGELIST_ICONS:
   {
      HIMAGELIST old, Icons;
      Icons = ImageList_Create(32, 32, ILC_COLOR4 | ILC_MASK, 4, 4);
      if (!Icons)
         break;
      if (!AddIcons(Icons)) {
         ImageList_Destroy(Icons);
         break;
      }
      old = TreeView_SetImageList(hwndTV, Icons, TVSIL_NORMAL);
      if (old)
         ImageList_Destroy(old);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_ICONS, MF_CHECKED);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_BITMAPS, MF_UNCHECKED);
      break;
   }
case ID_IMAGELIST_BITMAPS:
   {
      HIMAGELIST old, Bitmaps;
      Bitmaps = ImageList_Create(64, 64, ILC_COLOR4 | ILC_MASK, 4, 4);
      if (!Bitmaps)
         break;
      if (!AddBitmaps(Bitmaps)) {
         ImageList_Destroy(Bitmaps);
         break;
      }
      old = TreeView_SetImageList(hwndTV, Bitmaps, TVSIL_NORMAL);
      if (old)
         ImageList_Destroy(old);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_ICONS, MF_UNCHECKED);
      CheckMenuItem(GetMenu(hwnd), ID_IMAGELIST_BITMAPS, MF_CHECKED);
      break;
   }
```

Figure 183. TVTEST.C: Revised Code for ID_IMAGELIST_ICONS and ID_IMAGELIST_BITMAPS

Again, a preprocessor `#ifdef` is used to determine whether the OS/2 or Win32 function should be compiled. By isolating the system-dependent code into whole functions, the message processing code remains clear and easy to understand.

Since the functions are short and essentially the only system-dependent code in TVTEST.C, they have been left there. For larger implementations which use many images, it would probably be a better idea to separate the functions into two mixed mode dependent files. See Chapter 5, “Mixed Mode Sample Program” on page 133 for more information on mixed mode programming.

7.2.3 Converting Resources

You can use SMART to automatically convert the TVTEST.RC file from Win32 format to OS/2 format. SMART will also convert the Win32 format icon and bitmap files to OS/2 format. There should not be any errors during the

conversion, but you will need to make a few manual adjustments to the Resource Compiler file. First, you must add a statement at the beginning to include <os2.h>. Second, you must define IDC_STATIC, a constant defined in the Windows header files but not in the Open32 header files. Last, you need to comment out the line which includes afxres.h, since it is not needed for Open32 compilation. All of these changes are at the beginning of TVTEST.RC and are shown in Figure 184.

```
/* SM$FO J:\CD\TVTEST\OS2\tvtest.rc - Monday  08/19/1996  15:49:49 */
//Microsoft Developer Studio generated resource script.
//
#include <os2.h>
#include "tvtest.h"

#define IDC_STATIC -1

#define APSTUDIO_READONLY_SYMBOLS
////////////////////////////////////////////////////////////////////////
//
// Generated from the TEXTINCLUDE 2 resource.
//
//#include "afxres.h"
```

Figure 184. Adding <os2.h> to TVTEST.RC

After finishing the program and running it, you may notice that some dialog box items do not appear correctly because their text is cut off. This is due to differences in the font used in Windows and the one used in OS/2. You can either use the OS/2 dialog editor which comes with the OS/2 Warp Toolkit or manually edit the Resource Compiler file to change the items' widths.

7.2.4 Creating a Makefile

Before building the application, you will need to create a makefile for OS/2. The makefile from the OS/2 migrated TVTest sample program is shown in Figure 185 on page 221.

```
#
#  MAKEFILE:  OS/2 Makefile for TVTest
#

ALL: TVTEST.EXE

TVTEST.OBJ : TVTEST.C TVTEST.H
  ICC /I..\OPEN32TV /DOS2 /C /Ss TVTEST.C

MAIN.OBJ : MAIN.C
  ICC /DOS2 /C /Ss MAIN.C

TVTEST.RES: TVTEST.RC
  RC -r TVTEST.RC

TVTEST.EXE: TVTEST.OBJ TVTEST.RES ..\OPEN32\OPEN32TV.LIB MAIN.OBJ
  ILINK TVTEST.OBJ MAIN.OBJ PMWINX.LIB ..\OPEN32\OPEN32TV.LIB TVTEST.DEF
  RC TVTEST.RES TVTEST.EXE
```

Figure 185. MAKEFILE: TVTest Makefile

7.2.5 Creating a DEF File

The link definitions (DEF) file is required to give the application a name and a stack. The DEF file from the OS/2 migrated TVTest application is shown in Figure 186.

```
;
;  TVTEST.DEF:  Link definitions file for TVTest
;
NAME         TVTEST WINDOWAPI
DESCRIPTION   'TVTest Sample Application (C) Copyright IBM, Corp., 1996'
STACKSIZE    65536
```

Figure 186. TVTEST.DEF

7.2.6 Building the Application

After all the files are prepared, you need to build the application executable. Remember the link step must specify the OS/2 tree view library file to have it linked in with the application. This will resolve the call to InitTreeView() and provide the window procedures needed for the OS/2 tree view class. The makefile shown in Figure 185 already specifies the Open32TV library.

If you copied the files as recommended in 7.2.1, “Copying the Source Files” on page 213 and have modified them as outlined in this chapter, the program can be built as shown in Figure 187 on page 222.

```
[F:\CDIMAGE\TVTEST\OS2]nmake

Operating System/2 Program Maintenance Utility
Version 3.00.008 May  9 1995
Copyright (C) IBM Corporation 1988-1995
Copyright (C) Microsoft Corp. 1988-1991
All rights reserved.

    ICC /I..\OPEN32TV /DOS2 /C /Ss ..\TVTEST.C
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

H:\TOOLKIT\H\os2win.h(14:4) : warning EDC0523: Obsolete #pragma checkout ignored
. Use #pragma info or the /W options.
    RC -r TVTEST.RC
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Creating binary resource file TVTEST.RES
RC:  RCPP -E -D RC_INVOKED -W4 -f TVTEST.RC -ef H:\IBMCPP\BIN\RCPP.ERR -I H:\IBM
CPP\INCLUDE -I H:\IBMCPP\INCLUDE\OS2 -I H:\IBMCPP\INC -I H:\IBMCPP\INCLUDE\SOM -
I H:\TOOLKIT\BETA\H -I H:\TOOLKIT\SOM\INCLUDE -I H:\TOOLKIT\H -I H:\TOOLKIT\INC
-I F:\TOOLKIT\BETA\H -I F:\TOOLKIT\SOM\INCLUDE -I F:\TOOLKIT\H -I . -I F:\TOOLKI
T\INC

TVTEST.RC.................
    ICC /DOS2 /C /Ss MAIN.C
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

H:\TOOLKIT\H\os2win.h(14:4) : warning EDC0523: Obsolete #pragma checkout ignored
. Use #pragma info or the /W options.
    ILINK TVTEST.OBJ MAIN.OBJ PMWINX.LIB ..\OPEN32TV\OPEN32TV.LIB TVTEST.DEF

IBM(R) Linker for OS/2(R), Version 01.00.05
(C) Copyright IBM Corporation 1988, 1995.
(C) Copyright Microsoft Corp. 1988, 1989.
 - Licensed Material - Program-Property of IBM - All Rights Reserved.
```

Figure 187 (Part 1 of 2). Building TVTest

```
    RC TVTEST.RES TVTEST.EXE
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Reading binary resource file TVTEST.RES

.................
Writing resources to OS/2 v2.0 Linear .EXE file
Writing 1 DEMAND resource object(s)
  Writing:  18204 bytes in 5 page(s)
    108.1 (874 bytes)
    109.1 (874 bytes)
    110.1 (874 bytes)
    111.1 (874 bytes)
    112.1 (874 bytes)
    113.2 (2136 bytes)
    114.2 (2136 bytes)
    115.2 (2136 bytes)
    116.2 (2136 bytes)
    117.2 (2166 bytes)
    101.3 (603 bytes)
    102.4 (444 bytes)
    103.4 (774 bytes)
    104.4 (282 bytes)
    105.4 (438 bytes)
    106.4 (366 bytes)
    107.4 (194 bytes)


[F:\CDIMAGE\TVTEST\OS2]
```

Figure 187 (Part 2 of 2). Building TVTest

7.2.7 Running the New TVTest for OS/2

With the application fully built, you are ready to try the TVTest program under OS/2. After building the main executable, you can run the application from the command line as shown in Figure 188.

```
[F: CDIMAGE TVTEST OS2]tvtest
```

Figure 188. Running TVTest from the Command Line

The main window is shown in Figure 189 on page 224.

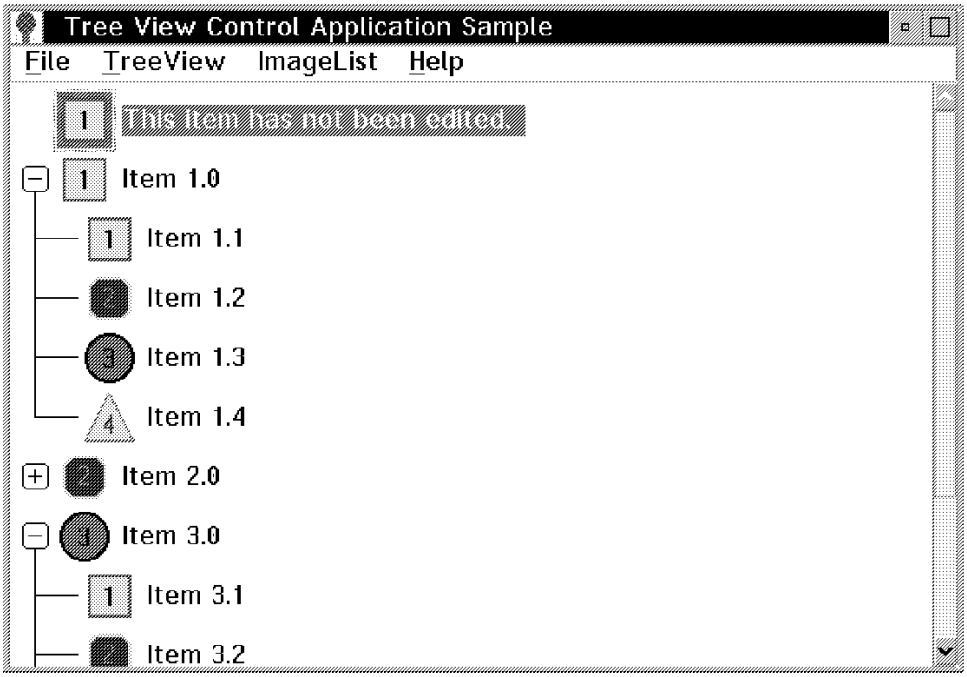


Figure 189. TVTest is now an OS/2 Program

7.3 Extending the OS/2 Tree View Control

The OS/2 tree view control used in this chapter is an incomplete translation control. While it provides the major functions required by most Win32 applications, it does not support some messages and functions. The following are not supported:

- Drag and drop
  - TVN_BEGINDRAG
  - TVN_BEGINRDRAG
  - TVM_CREATEDRAGIMAGE
  - TVGN_DROPHILITE
- Get edit control (TVM_GETEDITCONTROL)
- Hit testing (TVM_HITTEST)
- Sorting
  - TVM_SORTCHILDREN
  - TVM_SORTCHILDRENCB
- First visible item (TVGN_FIRSTVISIBLE)

- Incremental search strings (TVM_GETISEARCHSTRING)
- Information callback
  - TVN_GETDISPINFO
  - TVN_SETDISPINFO

For many applications, these unsupported messages are not used so the tree view translation control can be used without modification. For those applications that use these additional messages, the OS/2 tree view control can be extended to include support for them. The OS/2 container provides drag-and-drop, hit testing, sorting, and call-back functions, so these can be added to the translation control. An incremental search could be implemented by subclassing the container window so that keystroke messages can be trapped and used by the tree view translation control.

7.4 Creating your own Translation Controls

The technique used in this chapter to recreate the tree view control can be used to implement other Win32 window classes that are not supported by Developer API Extensions. To help you start migrating a window class, template files have been created with the basic functions needed to implement a Win32 control under OS/2. See 7.4.1, “Template Source Files” for a description of these files and their contents.

In developing a translation control, it is good idea to first write a Win32 test bed application such as TVTest. The tree view translation control was incrementally developed using the TVTest program as a test environment. TVTest gives you complete control over when and which messages are sent to the tree view control, letting you test different messages in different situations. One example would be deleting the last item in the tree view to make sure it does not attempt to access unallocated memory. TVTest also lets you test sending the tree view an invalid combination of parameters. An example would be inserting an item whose parent and previous sibling are the same item. By making the test bed application simple, it is much easier to debug the translation control.

7.4.1 Template Source Files

The template source files are on the CD in OPEN32CC. There are six files, as outlined in Table 11 on page 226.

Table 11. Files for a Custom Translation Control

File	Purpose
OPEN32CC.C	This file is compiled with the <os2win.h> header file and contains the Open32 source code for the translation control.
OS2CC.C	This file is compiled with the <os2.h> header file and contains the OS/2 source code for the translation control. This file must be modified the most, since most translation will occur in its two functions.
PRIVATE.H	This is the header file which contains private information on the structure of the translation control. It is compiled into OPEN32CC.C and OS2CC.C, but it will not be used by any applications which use the translation control.
OS2WINCC.H	This file contains the public declaration of the translation control and any associated constants, messages, functions, and data structures. It is distributed with the library and included by Open32 applications which use the translation control.
OS2CCTRL.H	This file contains general definitions needed by any translation control. It includes the NMHDR structure and NM_ message constants. It is generally best if only one copy of it is kept, since it also provides space for recording what translation controls use what notification message range constants.
MAKEFILE	An OS/2 makefile for building the source code into a library.

7.4.2 Modifying the Template

The first step in working with the template files is to create a copy of the files to work with. The files are designed to be used to create a library, which is subsequently used by a Win32 application. Therefore, the source files for the translation control are usually kept in a directory by themselves, instead of with the OS/2 dependent files for an application. The actual name of the directory dependents on your particular project and style.

7.4.2.1 Renaming Template Names

It is generally recommended that you rename the files to reflect the name of the control that they implement. As template files, the letters *CC* are used for *Common Control*. The letters are used throughout the templates,

including file names, function names, constants, and data structure names. To change the name, search for *CC* and replace occurrences with an acronym for your control. It is generally best to keep the acronym to just two letters, so that files such as OPEN32CC.C and OS2WINCC.H can retain their naming conventions.

7.4.2.2 Adding Code to the Templates

Their are several modifications which will probably have to be made before the translation controls is even minimally functional. First, you will need to add code to create the OS/2 equivalent control to the MsgToCC() function during WM_CREATE processing. The template already includes code to copy some information into the template's private data structure, such as the window handle of the parent.

The private data structure, named CCWNDDATA, will also need to be changed for most controls. Allocation of space for the structure and retrieval during message processing already occurs in the template.

After the new control properly processes the WM_CREATE message, you can incrementally add processing for control-specific messages. You should define message IDs in OS2WINCC.H, which is shared with the Win32 application. Don't compile the Win32 application with COMMCTRL.H, as the numeric IDs in it are not valid on OS/2 and may not match the IDs in OS2WINCC.H. Always begin numbering command messages with CCM_BASE, which is defined in OS2CCTRL.H.

Notification messages are sent with the WM_NOTIFY message to the Win32 application. As you add support for them, you should update the OS2CCTRL.H file to reflect the notification range you are using for the control. It is very important that the notification ranges for two controls do not overlap, as unpredictable results may occur in the application program.

7.4.2.3 How to Program Command Message Handling

Command messages are sent by the Win32 application to the translation control. Since the translation control was registered as a window class using the Win32 function RegisterClass(), messages from the Win32 application are processed by the Win32 window procedure CCOpen32WndProc(). You will need to modify this procedure by adding case statements for messages which your translation control will process. These should all be listed before a single call to MsgToCC(). An example of this, from the tree view control, is shown in Figure 190 on page 228.

```
LRESULT CALLBACK TVOpen32WndProc(HWND hwnd, UINT message,
                     WPARAM wParam, LPARAM lParam)
{
   . . .
   switch(message)
   {
      case WM_CREATE:
         pTVWndData = malloc(sizeof(TVWNDDATA));
         SetWindowLong(hwnd, 0, (long) pTVWndData);
         MsgToTV(hwnd, 0x0001, wParam, lParam, pTVWndData);  // WM_CREATE
         break;

      . . .

      case TVM_EXPAND:
      case TVM_GETCOUNT:
      case TVM_GETIMAGELIST:
      case TVM_GETINDENT:
      case TVM_GETITEM:
      case TVM_GETITEMRECT:
      case TVM_GETNEXTITEM:
      case TVM_SETINDENT:
      case TVM_SETITEM:
      case TVM_SETIMAGELIST:
         return MsgToTV(hwnd, message, wParam, lParam, pTVWndData);

      default:
         return DefWindowProc(hwnd, message, wParam, lParam);

   }
   return FALSE;
}
```

Figure 190. OPEN32TV.C: Directing Messages to MsgToTV()

The list of TVM_ constants causes those messages to be forwarded to MsgToTV(), which is defined in OS2TV.C.

While at first it might seem to make sense to translate command messages in CCOpen32WndProc(), doing so would actually require additional work. This is because during translation, the control must access both Win32 and OS/2 structures. The Win32 code, where CCOpen32WndProc() is written, has the Win32 structures defined, but not the OS/2 structures. The OS/2 code, where MsgToCC() is written, has the OS/2 structures defined, but not the Win32 structures. The key, however, is that you will already have to write the structures in OS2WINCC.H for the Win32 application. The OS/2 code can include OS2WINCC.H as well, and then have both the OS/2 and Win32 structures defined.

The MsgToCC() function will actually contain the code for translating all command messages. The contents of this function will depend on the control which you are implementing.

7.4.2.4 How to Program Notification Message Handling

Notification messages are sent by the OS/2 child control to the translation control interface. These need to be translated and forwarded to the Win32 parent to inform it of the user's actions with the control.

OS/2 calls CCOS2WndProc() to handle notification messages. This window procedure replaces the Win32 window procedure and is set up during the processing of WM_CREATE in MsgToCC(), shown in Figure 191.

```
LONG MsgToCC(HWND hwnd, ULONG msg, LONG wp, LONG lp, PCCWNDDATA pCCWndData)
{
   switch(msg)
   {
      case WM_CREATE:
      {
          PWINCREATESTRUCT cs;
          cs = (void *)lp;
          pCCWndData->hwndWinParent = cs->hwndParent;
          pCCWndData->WindowID = cs->hMenu;

          // Actually create the child here

          pCCWndData->oldWP=WinSubclassWindow(hwnd, (PFNWP) CCOS2WndProc);
          return FALSE;
      }
      . . .
```

Figure 191. OS2CC.C: WM_CREATE Processing

The code calls WinSubclassWindow() to make the OS/2 window procedure the function OS/2 will call when the child control sends messages to the translation control. Without this call, messages would go through the Open32 translation layer before they would be processed by CCOpen32WndProc(). The Open32 translation layer would not forward the control's notification messages because they are not supported by Open32.

When adding code to support notification messages, always call Open32SendMsg() to send a message to the parent window. If you use WinSendMsg(), the message will be removed and processed by Open32 support which is not what you want. You can access the parent window's handle by using pCCWndData->hwndParent.

7.4.3 Hints on Creating Translation Controls

In developing the technique described in this chapter, many variations on the general approach were tried, and essentially none of them worked. In this section we will outline some of the changes to the basic technique which seem like good ideas, but which actually don't work at all.

7.4.3.1 Don't Register the Window Class with WinRegisterClass()

Initially, we tried registering the translation window class directly with OS/2 because we knew we would need to receive messages from the OS/2 child control in an OS/2 window procedure. Unfortunately, Open32 apparently maintains an internal list of valid, registered window classes which it will allow Win32 applications to create. This allows Open32 to restrict Win32 applications from creating windows whose messages Open32 does not know how to translate. Unfortunately, it also prevents Win32 applications from creating OS/2-registered windows using CreateWindow().

The solution, as used by the tree view control and the template for other common controls, is to register the class with Open32 using RegisterClass(). During WM_CREATE processing, WinSubclassWindow() is called to replace the Win32 window procedure with an OS/2 window procedure. Messages from the child control will be sent to the OS/2 window procedure. Open32 will continue to send messages from the Win32 application to the Win32 window procedure because it maintains internally the address of the window procedure which was specified when the window class was registered using RegisterClass().

7.4.3.2 Don't Send a Message to Yourself with WinSendMsg()

When a translation control needs to send itself a command message, don't use WinSendMsg(). For example, in the tree view control's processing of the TVM_SETIMAGELIST message, it adjusts the container's indent to the size of the images. To do this, it sends itself the TVM_SETINDENT message using Open32SendMsg(). If WinSendMsg() were used, the message would be sent to TVOS2WndProc(), which does not handle the TVM_SETINDENT message.

7.4.3.3 Don't Use Open32 Image Handles

Some Win32 common controls use icons and bitmaps to enhance their displays. These images complicate the creation of a translation control because image handles returned by Open32 functions cannot be used by standard OS/2 functions. Therefore, images which the Win32 application loads cannot be passed to OS/2 controls, such as the container.

In the tree view example, this problem was handled by replacing the code which loads the images with a platform-dependent function. When compiled for OS/2, the function passes the resource ID of the image to the Open32 image list, which then loads the image using OS/2 functions.

This approach has one major advantage over any other method: it is relatively simple to implement. There are, however, two drawbacks. First, the resources must be bound in the same executable or DLL as the Open32

image list functions. Second, the Win32 program must be changed to pass the resource ID instead of the image handle.

In the tree view example, the first drawback was not a problem because all of the code and resources were stored in a single executable. The second drawback was also minor because the code for dealing with images was already encapsulated by the image list, so changes to the Win32 program could be isolated to a single function.

Large applications will expose the limitations of this approach to image handling. In large applications, much of the application's program code is in separate DLLs, and resources may be either bound to the DLL that accesses them or bound together in a single resource DLL. Large applications may also access more images throughout the program, stipulating greater changes to the Win32 source code.

There is not easy solution to the limitations introduced by the simple design of the Open32 image list. One possible approach to resolving the problem is to allow the Win32 application to load the resources itself. This will eliminate both limitations outlined above.

However, elaborate owner-draw code must be added to the translation control to draw the images using Win32 function calls. The container, for example, would send a CN_OWNDERDRAW message to CCOS2WndProc(), which would call a custom procedure in OPEN32CC.C to draw the image in the correct location. Due to the complexity of this approach, it has not been attempted and its feasibility is unknown.

7.4.3.4 Don't Change the Win32 Test Application

In developing a translation control, it is very important that you not change the Win32 test bed application which you are using to test the translation control. This may seem obvious (after all, the whole point of a translation control to is avoid changing the program), but it can be very tempting to make a slight modification to the program so it will work properly with the translation control. You should make every effort to adapt the translation control to the Win32 program, and not the other way around.

This rule includes changes to the program's header files. You should design the application so that it includes only one header file for each common control it uses. You may need to adapt data types used in the header file. Figure 192 on page 232 shows a portion of OS2WINTV.H which defines PRECORDCORE only when compiling Win32 code.

```
/* If compiling under Open32, define PRECORDCORE as pointer.      */
#ifdef _OS2WIN_H
typedef struct _RECORDCORE RECORDCORE, *PRECORDCORE;
#endif

typedef PRECORDCORE HTREEITEM;
```

Figure 192. OS2WINTV.H: Conditional Definition of PRECORDCORE

If _OS2WIN_H is defined, then the <os2win.h> file has been included. Therefore, the PRECORDCORE structure defined in <os2.h> is undefined. We define it simply as a pointer to a structure. By conditionally defining PRECORDCORE, we can be sure that it is defined before the typedef statement which declares the type HTREEITEM.

7.4.3.5 Don't Use WinGetWindowULong()

You cannot use the OS/2 function WinGetWindowULong() to retrieve the window data for the translation control. The control is registered through Open32, and Open32 stores the window data. You must use the Win32 function GetWindowLong() to access the data. If you need to access the data in an OS/2 compiled file (such as OS2CC.C), use Open32GetLong(), which is defined in GENERAL.C.

Note

If you need to store window data, use the window data structure already defined for you in PRIVATE.H. See 7.4.3.7, “Use Window Data for Reentracy Compatibility” for more information.

7.4.3.6 Update OS2CCTRL.H

Whenever you begin to translate a control, update OS2CCTRL.H to claim the notification message IDs your control will use. This file should be shared by everyone who uses or writes translation controls. If more than one copy exists, notification message IDs may overlap and cause errors in the Win32 program.

7.4.3.7 Use Window Data for Reentracy Compatibility

If you need to store window data, use the window data structure already defined for you in PRIVATE.H. You can access members of the structure with pCCWndData-> in both MsgToCC() and CCOS2WndProc(). These two functions already contain code to load the pointer to the data when they are called. Memory for the structure is allocated in CCOpen32WndProc() when it processes the WM_CREATE message.

You should avoid using static variables in MsgToCC() and CCOS2WndProc() because they can cause reentrancy problems. If more than one translation control is created at once, the two controls will overwrite each other's data in the static variables. This will cause display problems for one or both controls and may cause the program to crash. Always store data in the CCWNDDATA structure.

Chapter 8. Existing Windows 16-bit Application Ported to OS/2

This section describes the migration of an existing Windows 16-bit application to the OS/2 environment.

When porting a Windows 16-bit application to OS/2, there are many complex issues which you will be faced with. While Open32 simplifies the migration of the application's window interface, it does not address some problems such as:

- Adjusting to the 32-bit flat memory model
- Replacing calls to DOS and BIOS interrupts for system services with calls to Win32 equivalents
- Migrating on-line help files
- Adapting the program when previously used APIs are no longer supported or no equivalent is available
- Supporting long file names

In this chapter we will discuss the steps required to port an existing Windows 16-bit application to OS/2 using Open32 and how we address the issues that we discovered in the approach we used.

The Address Manager sample program is used to demonstrate the steps required to migrate an existing Windows 16-bit application program. A copy of the original Win16 source code can be found in the ADDRESS WIN16 directory of the CD-ROM in this redbook. This code is redistributed from *Mastering Windows Utilities Programming with C++* by Michael J. Young, by permission of SYBEX Inc. ISBN 0-7821-1286-2, Copyright 1994, SYBEX Inc. All rights reserved.

We chose an existing Windows 16-bit application to migrate instead of writing a new Windows 16-bit application sample in hopes of better demonstrating the issues that you will face when porting your Windows 16-bit application programs and the steps required to migrate the application to Open32.

8.1 Overview of the Program Structure

The Address program is written in C++ using a simple class library (it does not use the Microsoft Foundation Class Library). The structure of the library is shown in Figure 193 on page 236.

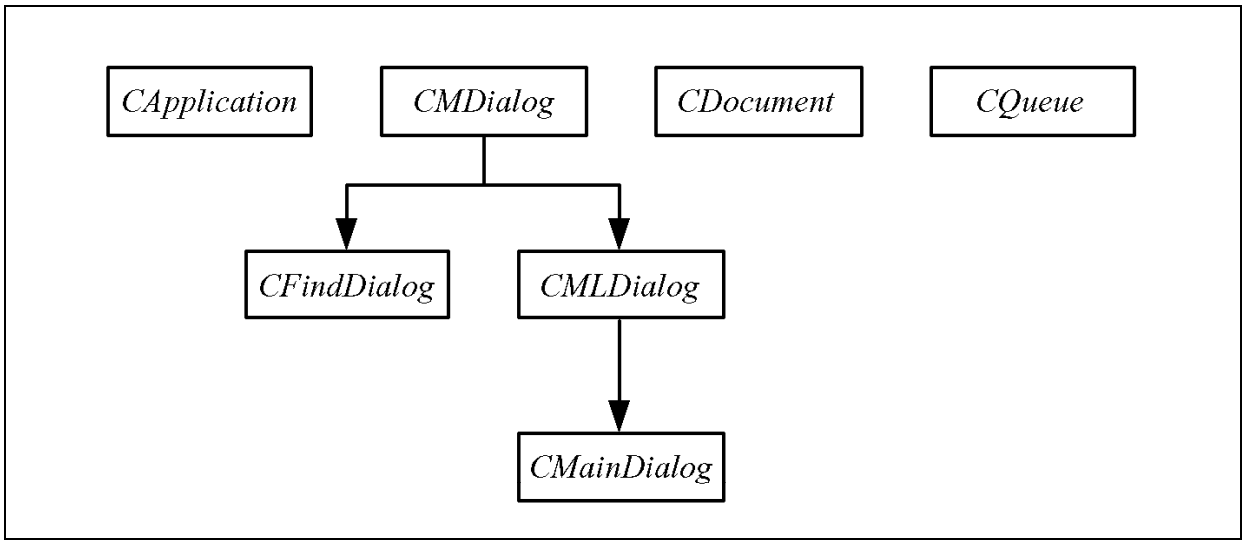


Figure 193. Window Class Structure Used in Address

The CApplication class encapsulates some of the general Windows programming details. It is declared once as a global variable for a program. Address Manager calls CApplication::Initialize() to set the commonly accessed variables hInstCurrent, hInstPrevious, CmdLine, and CmdShow. These are the same variables passed to the program's WinMain() procedure.

CMDialog encapsulates the details of creating and managing a model dialog box. Address Manager subclasses CMDialog into CFindDialog, which manages a search dialog.

CMLDialog encapsulates the details of creating and managing a modeless dialog box. Address Manager subclasses CMLDialog into CMainDialog, which manages the program's main window.

Address also defines CDocument, which handles the details of file management, and CQueue, which stores data in a linked list.

8.2 Overview of the Migration Process

The first step is to copy the Win16 source files to a new directory where they can be modified and overwritten during the migration process. The Win16 source files are on the CD in the directory ADDRESS WIN16. A completely ported application is in the ADDRESS OS2 directory.

```
[D:\OPEN32]md ADDRESS

[D:\OPEN32]md ADDRESS\MIGRATE

[D:\OPEN32]cd ADDRESS\MIGRATE

[D:\OPEN32\ADDRESS\MIGRATE]XCOPY F:\ADDRESS\WIN16\*

Source files are being read...

F:\ADDRESS\WIN16\HELP.RTF
F:\ADDRESS\WIN16\ADDRESS.HLP
F:\ADDRESS\WIN16\ADDRESS.EXE
F:\ADDRESS\WIN16\DIALOG.CPP
F:\ADDRESS\WIN16\ADDRESS.IDE
F:\ADDRESS\WIN16\ADDRESS.HPJ
F:\ADDRESS\WIN16\DOCUMENT.CPP
F:\ADDRESS\WIN16\ADDRESS.CPP
F:\ADDRESS\WIN16\ADDRESS.H
F:\ADDRESS\WIN16\ADDRESS.RC
F:\ADDRESS\WIN16\RESOURCE.H
F:\ADDRESS\WIN16\ADDRESS.DEF
F:\ADDRESS\WIN16\HELP02.BMP
F:\ADDRESS\WIN16\HELP01.BMP
F:\ADDRESS\WIN16\ADDRESS.MAK
F:\ADDRESS\WIN16\ADDRESS.ICO
F:\ADDRESS\WIN16\WCLASSES.CPP
F:\ADDRESS\WIN16\WCLASSES.H

18 file(s) copied.

[D:\OPEN32\ADDRESS\MIGRATE]
```

Figure 194. Copying the Win16 Source Files for Migration

We recommend that you copy the files from ADDRESS WIN16 to your hard drive in the directory OPEN32 ADDRESS MIGRATE. This process is shown in Figure 194.

8.2.1 Changes to the Source Code

After the files have been copied, you need to modify the source code to be Open32 compatible. There are quite a few changes to be made because the

application is effectively going through two changes at once. First, it is becoming Win32 compatible, and second it is becoming Open32 compatible.

Table 13 describes all of the changes that are needed to make the source files compile and run under OS/2. The changes must be made for a variety of reasons. As a result, the cause of each change is marked in Table 13 in the Reason column. An explanation of the change codes is given in Table 12.

Table 12. Codes for Changes to Address

Code	Explanation
O	Change made because of a difference between Open32 and Win32.
W32	Change made because of a difference between Win32 and Win16.
C	Change made because of a difference between Microsoft Visual C++ and VisualAge C++.

Table 13 (Page 1 of 4). Changes to the Address Source Files

File	Line	Reason	Description of change

wclasses.h	9	O	Replace
			`#include <windows.h>`
			with
			`#include <os2win.h>`

wclasses.h	77	W32	Remove "_export" from the definition of GenDlgProc().
address.h	78	W32	Remove "far" from the definition of FileOpen().
address.cpp	31-44	W32	Remove these lines. There is no Win32 equivalent for activating a previous instance of a program.
dialog.cpp	9	O	Remove the line `#include <commdlg.h>` because common dialogs are standard in <os2win.h>.

dialog.cpp	41	W32	Replace
			`return OnCommand ((int)WParam, (HWND)LOWORD (LParam), HIWORD (LParam));`
			with its Win32 equivalent
			`return OnCommand (LOWORD(WParam), (HWND) (LParam), HIWORD (WParam));`

dialog.cpp	44	W32	Replace WM_CTLCOLOR with WM_CTLCOLORDLG.

Table 13 (Page 2 of 4). Changes to the Address Source Files

File	Line	Reason	Description of change

dialog.cpp	256	O	Replace
			`HWnd = FindWindow (0, "Phone Dialer");`
			with
			`HWnd = FindWindow ("", "Phone Dialer");`

dialog.cpp	274	O	Replace
			`HWnd = FindWindow (0, "Phone Dialer");`
			with
			`HWnd = FindWindow ("", "Phone Dialer");`

dialog.cpp	446	O	Replace
			`HWnd = FindWindow (0, "Envelope & Label Printer");`
			with
			`HWnd = FindWindow ("", "Envelope & Label Printer");`

dialog.cpp	464	O	Replace
			`HWnd = FindWindow (0, "Envelope & Label Printer");`
			with
			`HWnd = FindWindow ("", "Envelope & Label Printer");`

dialog.cpp	550	O	Insert the following lines before the call to GetPrivateProfileInt():
			`HMENU mHMenu;`
			`mHMenu = LoadMenu(App.mHInstCurrent, "ADDRESSMENU");`
			`SetMenu(mHDialog, mHMenu);`

dialog.cpp	757	W32	Replace
			`switch(WParam)`
			with its Win32 equivalent
			`switch(LOWORD(WParam))`

dialog.cpp	765	W32	Replace
			`if (HIWORD (LParam) == EN_CHANGE)`
			with its Win32 equivalent
			`if (HIWORD (WParam) == EN_CHANGE)`

wclasses.cpp	18	W32	Remove “_export” from the definition of GenDlgProc().
wclasses.cpp	76	W32	Remove “_export” from the definition of MessageProc().

Table 13 (Page 3 of 4). Changes to the Address Source Files

File	Line	Reason	Description of change

wclasses.cpp	167	W32	Replace
			`GetCurrentTask ()`
			with its Win32 equivalent
			`GetCurrentThreadId ()`

wclasses.cpp	177	W32	Remove `far` from the type cast of `this`.
wclasses.cpp	230	W32	Remove `far` from the type cast of `this`.
document.cpp	22	W32	Remove the line `_fmode = O_BINARY;`
document.cpp	109-110	W32	Replace the call to OpenFile() with `DeleteFile(TempFileStruct.szPathName);`
document.cpp	113	C	Replace the definition of TempFileName with `char *TempFileName;`

document.cpp	114	C	Replace
			`mktemp (TempFileName);`
			with
			`TempFileName = tmpnam(NULL);`

document.cpp	118-121	W32	Replace the call to OpenFile() with
			`HTempFile = open(TempFileName, O_CREAT | O_RDWR | O_BINARY, S_IWRITE);`
			`strcpy(TempFileStruct.szPathName, TempFileName);`

document.cpp	128	W32	Remove "_far" from the declaration of CDocument::FileOpen.

document.cpp	137-140	W32	Replace the call to OpenFile() with
			`HFILE HPermFile = open(FileName, O_RDONLY | O_BINARY, S_IWRITE);`
			`strcpy(PermFileStruct.szPathName, FileName);`

document.cpp	154-155	W32	Replace the call to OpenFile() with `DeleteFile(TempFileStruct.szPathName);`
document.cpp	158	C	Change the definition of TempFileName to `char *TempFileName;`

document.cpp	159	C	Replace the line
			`mktemp (TempFileName);`
			with
			`TempFileName = tmpnam(NULL);`

Table 13 (Page 4 of 4). Changes to the Address Source Files

File	Line	Reason	Description of change

document.cpp	163-166	W32	Replace the call to OpenFile() with
			`HTempFile = open(TempFileName, O_CREAT | O_RDWR | O_BINARY, S_IWRITE);`
			`strcpy(TempFileStruct.szPathName, TempFileName);`

document.cpp	177	W32	Replace the call to _lclose() with `close (HPermFile);`
document.cpp	211-212	W32	Replace the call to OpenFile() with `HPermFile = open(PermFileStruct.szPathName, O_CREAT | O_WRONLY | O_BINARY, S_IWRITE);`
document.cpp	216-217	W32	Replace the call to OpenFile() with `HPermFile = open(FileName, O_CREAT | O_WRONLY | O_BINARY, S_IWRITE);`
document.cpp	258	W32	Replace the call to _lclose() with `close (HPermFile);`

document.cpp	605-606	W32	Replace the lines
			`GetDlgItemText (MainDialog.mHDialog, IDC_PHONES, Phones, Size);`
			with
			`if (Size==1) {`
			`  *Phones = 0;`
			`} else {`
			`  GetDlgItemText (MainDialog.mHDialog, IDC_PHONES, Phones, Size);`
			`} /* endif */`

document.cpp	615	W32	Replace the line
			`GetDlgItemText (MainDialog.mHDialog, IDC_NOTES, Notes, Size);`
			with
			`if (Size==1) {`
			`  *Notes = 0;`
			`} else {`
			`  GetDlgItemText (MainDialog.mHDialog, IDC_NOTES, Notes, Size);`
			`} /* endif */`

document.cpp	644-645	W32	Replace the call to OpenFile() with `DeleteFile(TempFileStruct.szPathName);`

8.2.1.1 General Changes

Several of the changes involve simply removing `far` or `_export` from the definition of a function or type cast. Both of these changes are standard in moving a Windows 16-bit application to Win32.

8.2.1.2 Changes to ADDRESS.CPP

Lines 31 through 44 of the original source are executed only if a previous instance of the Address Manager is already running in the system. It uses the Win16 function GetInstanceData(), for which there is no apparent Win32 equivalent. Without any way to retrieve the window handle of the previous instance, the program cannot bring it to the top of the z-order. As such, it cannot ensure that only one instance of the program is running. By removing all of lines 31 through 44, the program changes to allow numerous instances of the program to run at once.

8.2.1.3 Changes to DIALOG.CPP

The first group of changes to DIALOG.CPP deal with changes to the use of `wParam` and `lParam` between Win16 and Win32. Microsoft changed wParam and lParam for Win32 because of two other changes involved in moving to a 32-bit operating system:

- The size of HWND was increased to 32 bits.
- The size of WPARAM was increased to 32 bits (a WORD is now the same size as a long integer).

The second group of changes concerns the four calls to FindWindow(). The first parameter of the function call specifies the class name, if any, of the window to locate. It is permissible to pass zero to the function under Win32, but not under Open32. Open32 apparently attempts to access the memory location pointed to by the string pointer before validating the address. Therefore, passing zero causes Open32 to try to access memory location 0x00000000, which is reserved. By passing a null string, FindWindow() will access a valid piece of memory.

On line 550, three lines of code need to be inserted to set the dialog's menu. While Windows allows you to specify the menu for a dialog box in the Resource Compiler file, OS/2 requires that a menu be attached to a dialog during execution. For this reason, the menu is loaded and set during program initialization by these three lines of code.

8.2.1.4 Changes to DOCUMENT.CPP

Many of the changes in DOCUMENT.CPP deal with replacing the OpenFile() function. While Open32 supports the standard function OpenFile(), it unfortunately supports the call exactly as Win32 does. Neither system honors the _fmode global variable to determine whether files should be

opened in binary or text mode. The Open32 include files do not even define the variable, which causes a compiler error to be generated on line 22 of DOCUMENT.CPP. Under Windows NT or Windows 95, Microsoft Visual C++ will compile the code without error, but the program will not function correctly.

Since the address program must access files using binary mode, the calls to OpenFile() must be changed. While the Win32 function CreateFile() could be used, using it would require extensive changes to the file because its file handle cannot be used with read() or write(). Files opened with CreateFile() must be accessed using ReadFile() and WriteFile().

The decision was made to instead use the standard C library function open(). Since open() does not use the FILESTRUCT structure, the filename must be copied to it using strcpy(). All of these changes are noted in Table 13 on page 238.

8.2.2 Converting the Resource Compiler File

After the source files are modified, you should convert the Resource Compiler file to OS/2 format using SMART. Be sure to specify Support String ID.

After SMART converts the Resource Compiler file, you will still need to make a small change to the resource script. The About dialog box contains the application's icon. This is a common practice in About dialog boxes for Windows applications. The problem, however, is that OS/2 cannot load a dialog box which contains an icon with a string ID. For this reason, the icon cannot have a string ID.

There are three possible solutions to this problem:

1. Duplicate the definition of the icon in the resource script and give it a numeric ID which is used in the dialog box. This, however, will create two identical copies of the icon in the application's executable file, wasting approximately 1 kilobyte.
2. Give the icon a numeric ID and change the source code when it references the icon. While this option eliminates the duplicate icon, it requires a change to the source code and leaves the application in something of a resource ID limbo; some identifiers are string, some are numeric.
3. Reconvert the resource script using SMART, only this time without Support String ID. When SMART finds a string ID in the resource file, it will automatically assign a numerical identifier to it and put the definition in a .HHH file. You can then include the .HHH file in the program and

use the new numeric IDs. However, this requires that you modify every line of code which references a resource.

For an example program like Address, it is easiest to just duplicate the icon and change the reference in the About box definition (Option 1 above). For major applications, it may be better to invest the time and effort of converting all string IDs to numeric IDs (option 3 above).

To change the Resource Compiler file, duplicate line 152 in ADDRESS.RC and change ADDRESSICON to a number, as shown in Figure 195.

```
147: ////////////////////////////////////////////////////////////
148: //
149: // Icon
150: //
151:
152: ICON           "ADDRESSICON"   DISCARDABLE    "ADDRESS.ICO"
153: ICON           1001           DISCARDABLE    "ADDRESS.ICO"
```

Figure 195. ADDRESS.RC: Changes to ADDRESSICON

After you duplicate the icon statement, change the definition of the About box, ABOUTDLG, to use the second icon. Edit line 99, as shown in Figure 196.

```
 91: DLGTEMPLATE "ABOUTDLG" DISCARDABLE
 92: BEGIN
 93:   DIALOG "About", -1, 15, 183, 145, 100,
 94:   WS_VISIBLE | FS_DLGBORDER | FS_SCREENALIGN,
 95:   FCF_TITLEBAR | FCF_SYSMENU | FCF_NOMOVEWITHOWNER
 96:   BEGIN
 97:     CTEXT       "Windows Address Manager",-1, 0,  82, 145,   8, SS_TEXT |
                  DT_WORDBREAK | DT_MNEMONIC
 98:     CTEXT       "by Michael J. Young",  -1,   0,  64, 145,   8, SS_TEXT |
                  DT_WORDBREAK | DT_MNEMONIC
 99:     ICON        1001,                   -1,  60,  36,  20,  16
100:     DEFPUSHBUTTON "OK",            DID_OK,  45,   6,  50,  14
101:   END
102: END
```

Figure 196. ADDRESS.RC: Changes to ABOUTDLG

You should also modify the definition of the main dialog, MAINDLG, to include FCF_TASKLIST in its series of FCF_ constants. This will automatically add the Address Manager to the task list while it is running. The dialog height also needs to be adjusted to allow room for the menu at the top of the dialog. A proper height is 195 dialog units, as highlighted in Figure 197 on page 245.

```
43: DLGTEMPLATE "MAINDLG" DISCARDABLE
44: BEGIN
45:   DIALOG "Address Manager", -1, 0, 122, 222, 195,
46:   WS_VISIBLE | FS_BORDER | FS_SCREENALIGN,
47:   FCF_TITLEBAR | FCF_SYSMENU | FCF_MINBUTTON | FCF_NOMOVEWITHOWNER | FCF_TASKLIST
48:   BEGIN
```

Figure 197. Changes to ADDRESS.RC

8.2.3 Converting Graphical Resources

After the Resource Compiler file is converted, you should use SMART to convert the graphical resources for Address. The program's resource file only has one icon, ADDRESS.ICO, to convert. However, there are also two bitmaps, HELP01.BMP and HELP02.BMP, which need to be converted before the help file can be translated. You should use SMART to convert all three files to OS/2 format. See 3.1.4, “Converting Resource Compiler Files” on page 83 for information on using SMART to convert icons and bitmaps.

8.3 Converting the Help File

You also need to convert the program's help file. The Windows version of the help file is in RTF format, which SMART can convert into OS/2's IPF format. From the SMART main window, select **Translate Win Help...**, as shown in Figure 198.

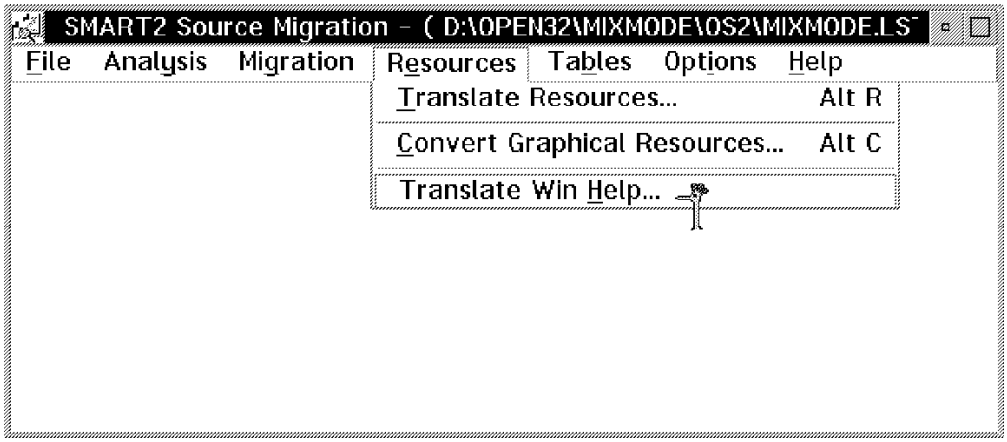


Figure 198. Selecting "Translate Win Help..." in SMART

The Win Help Translator dialog box will appear, as shown in Figure 199 on page 246. Use the **Select...** pushbuttons to specify the source and target

files. The output of the translator will be an OS/2 binary .HLP file. If you want the source IPF file to be created also, select the **Do Not Delete IPF file** check box.

After you have select the input and output files, select the **OK** button and SMART will convert the help file.

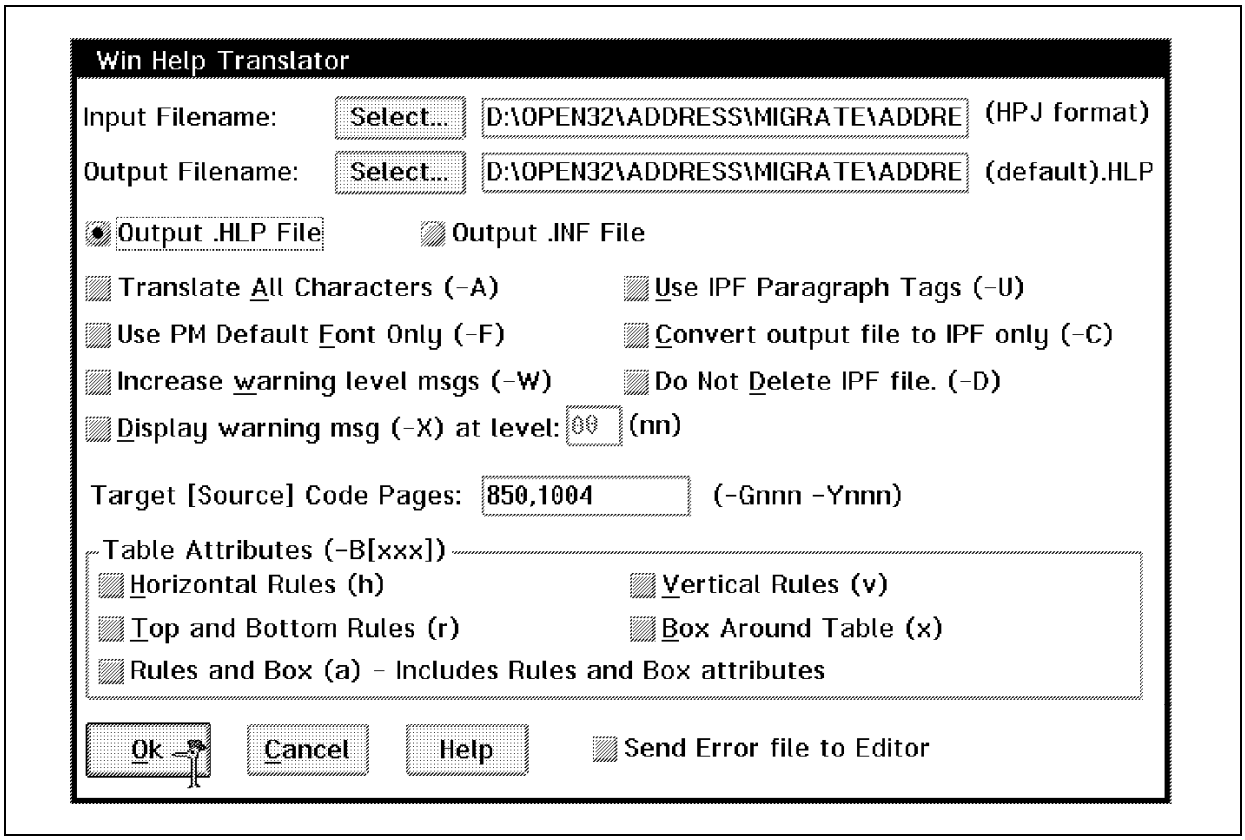


Figure 199. Win Help Translator Dialog Box

8.3.1 Creating a New Makefile

The process of recompiling the source files and resources will be made much easier by using NMAKE, the OS/2 Program Maintenance Utility. A makefile for the converted program is shown in Figure 200 on page 247.

```
#
#  MAKEFILE:  for Address Manager sample
#
ALL: address.exe

.rc.res:
   rc.exe -r %s

.CPP.obj:
   icc.exe /Gh /Ti /Tm /C %s

.C.obj:
   icc.exe /Ss /Gh /Ti /Tm /C %s

address.exe: address.obj dialog.obj document.obj \
          wclasses.obj main.obj address.res
   ilink /DEBUG address.obj main.obj document.obj \
            wclasses.obj dialog.obj pmwinx.lib address.def
   rc.exe address.res address.exe

address.res: address.rc resource.h
document.obj: document.cpp address.h wclasses.h resource.h
dialog.obj: dialog.cpp address.h wclasses.h resource.h
address.obj: address.cpp address.h wclasses.h resource.h
wclasses.obj: wclasses.cpp wclasses.h
main.obj: main.c
```

Figure 200. Makefile for OS/2

8.3.2 Creating the DEF File

A link definitions file is required by ILINK to define the application type and stack size. A sample DEF file is shown in Figure 201. You should create the file before building the application.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                      ;
; ADDRESS.DEF:  Module-definition file for the ADDRESS        ;
;             program.                                          ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

NAME ADDRESS WINDOWAPI
DESCRIPTION  'Address Manager'
STACKSIZE 65536
```

Figure 201. ADDRESS.DEF

8.3.3 Creating the Executable

After the source code and resources have been adapted to OS/2, you are ready to run NMAKE and build the application's executable file. Figure 202 on page 248 shows the output of the build process.

```
[D:\open32\address\migrate]nmake

Operating System/2 Program Maintenance Utility
Version 3.00.008 May  9 1995
Copyright (C) IBM Corporation 1988-1995
Copyright (C) Microsoft Corp. 1988-1991
All rights reserved.

    icc.exe /Gh /Ti /Tm /C address.CPP
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

    icc.exe /Gh /Ti /Tm /C dialog.CPP
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

    icc.exe /Gh /Ti /Tm /C document.CPP
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

    icc.exe /Gh /Ti /Tm /C wclasses.CPP
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

    icc.exe /Ss /Gh /Ti /Tm /C main.C
IBM VisualAge C ++ for OS/2, Version 3
(C) Copyright IBM Corp. 1991, 1995.
- Licensed Materials - Program Property of IBM - All Rights Reserved.

H:\TOOLKIT\H\os2win.h(14:4) : warning EDC0523: Obsolete #pragma checkout ignored
. Use #pragma info or the /W options.
    rc.exe -r address.rc
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Creating binary resource file address.RES
RC:  RCPP -E -D RC_INVOKED -W4 -f address.rc -ef H:\IBMCPP\BIN\RCPP.ERR -I H:\IB
MCPP\INCLUDE -I H:\IBMCPP\INCLUDE\OS2 -I H:\IBMCPP\INC -I H:\IBMCPP\INCLUDE\SOM
-I H:\TOOLKIT\BETA\H -I H:\TOOLKIT\SOM\INCLUDE -I H:\TOOLKIT\H -I H:\TOOLKIT\INC
 -I F:\TOOLKIT\BETA\H -I F:\TOOLKIT\SOM\INCLUDE -I F:\TOOLKIT\H -I . -I F:\TOOLK
IT\INC
```

Figure 202 (Part 1 of 2). Output of NMAKE During Application Build

```
address.rc........
       ilink /DEBUG address.obj main.obj document.obj  wclasses.obj dialog.obj
pmwinx.lib address.def

IBM(R) Linker for OS/2(R), Version 01.00.05
(C) Copyright IBM Corporation 1988, 1995.
(C) Copyright Microsoft Corp. 1988, 1989.
 - Licensed Material - Program-Property of IBM - All Rights Reserved.

    rc.exe address.res address.exe
Operating System/2 Resource Compiler
Version 3.01.002 Mar 12 1996
(C) Copyright IBM Corporation 1988-1996
(C) Copyright Microsoft Corp. 1985-1996
All rights reserved.


Reading binary resource file address.res

Writing resources to OS/2 v2.0 Linear .EXE file
Writing 2 DEMAND resource object(s)
  Writing:  4060 bytes in 1 page(s)
    302.1 (874 bytes)
    1001.1 (874 bytes)
    63788.3 (521 bytes)
    70.4 (299 bytes)
    60153.4 (224 bytes)
    64825.4 (250 bytes)
    64835.4 (983 bytes)
    65002.8 (22 bytes)
  Writing:  120 bytes in 1 page(s)
    8.255 (21 bytes)
    4.255 (53 bytes)
    3.255 (20 bytes)
    1.255 (20 bytes)
Writing 681 bytes of module format directive

[D:\open32\address\migrate]
```

Figure 202 (Part 2 of 2). Output of NMAKE During Application Build

8.3.4 Running Address

After the executable is built and the help file is converted, you are ready to run the new copy of Address for OS/2. The main Address window is shown in Figure 203 on page 250 running under OS/2 Warp Version 4.

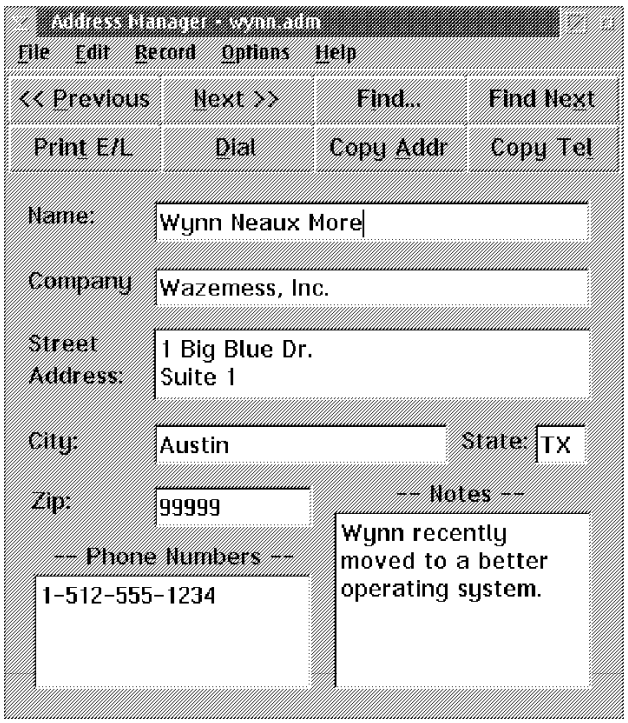


Figure 203. Address Main Dialog under OS/2

Note

As compiled, Address will not run under OS/2 Warp with the Developer API Extensions from The Developer Connection for OS/2 Volume 10, due to bugs in Release 1.0 of Developer API Extensions. The screen shot in Figure 203 is from OS/2 Warp Version 4. To run Address on OS/2 Warp with Developer API Extensions from The Developer Connection for OS/2 Volume 10, you must comment out all of the lines pertaining to Address's private profile data. A copy of the Address program with these changes made is provided on the CD accompanying this book in the ADDRESS WARP3 directory.

Chapter 9. Hints and Tips for Open32

This chapter describes several items to consider when designing, developing and coding applications from a common source base for execution in both the OS/2 and Win32 environments.

Unfortunately, many applications use at least one feature which is unsupported by Open32. In the rest of this chapter, we will discuss some of the lessons we learned in developing the sample applications for this redbook.

9.1 General Design Hints and Tips

The cost of developing and maintaining advanced graphical applications is often very high. Attempting to maintain an application on two platforms only adds to the cost (although this is usually offset by the increased market). Open32 greatly reduces the cost of dual-platform development and maintenance, and by judicious planning the cost can approach zero. The following general guidelines will result in decreased cost:

- Maximize the common code
- Minimize the dependent code
- Put general program code in the common code and interface details in the dependent code. Changes to the program are thus more likely to be in common code.
- Use low-maintenance techniques, such as the translation control (see Chapter 7, “Tree View Control Sample Program” on page 195).

9.1.1 New Program Design Hints & Tips

Windows 32-bit applications are easiest to migrate using Open32 when they use only the supported APIs and window classes. They require practically no changes to the source code, and the resources need only to be converted by SMART. In truth, any application, which uses only supported APIs, window classes, and resources can be migrated to OS/2 as easily as Howdy, regardless of size (See Chapter 3, “Howdy, World!” on page 79).

Therefore, the key to writing new Windows 32-bit applications for migration using Open32 is to maximize the common source code. If you do need to use common controls or unsupported APIs, early planning can save you a lot of time later in the project. When designing a new application which uses Win32 features that are not supported by Open32, keep the following in mind:

1. Identify unsupported features as early in the design as possible.
2. Decide how they will be accomodated.
   - Mix mode programming is generally more flexible (it allows features in OS/2 but not Win32 to be exploited), but it is also somewhat more cumbersome to use. The continuing development costs of mix mode programming are also higher than translation programming because two separate source codes must be maintained.
   - In comparison, translation programming allows an application to remain entirely common source but requires that OS/2 offer a similar feature to Win32, and it restricts the use of the OS/2 features to what Win32 offers. If, however, a complete translation feature already exists (for example, from a previous project), then it is probably the best solution. Reusing a translation feature costs essentially nothing, but writing a new one can be a substantial investment, depending on the complexity of the feature.
3. Decide how cross-platform help will be maintained before writing any of it.
   - IBM's Hyperwise allows the creation of a common help source for both Windows and OS/2 applications. However, its support for Windows help is through a special interface, not through the standard Windows help interface.
   - SMART offers an RTF-to-HLP conversion, allowing the help to be maintained for Windows and converted for OS/2. However, Open32 does not support all of the `WinHelp()` commands.

9.1.2 Existing Program Migration Hints and Tips

Existing programs can take only minimal effort or a substantial investment to migrate, depending on their current structure and feature use. Generally, the more features a program uses which are not directly supported by Open32, the more work it will take to migrate.

Mix mode programming is usually not a viable option for an existing application because it usually requires a redesign of the application's coding structure. Any code which interacts with the unsupported features must be extracted from the common code and isolated in dependent files.

We recommend translation programming for existing applications wherever it is available. Translation programming lets the application's primary source code remain intact while a special interface layer between it and OS/2 is added. The drawback with translation programming is that writing the translation code can be expensive because the programmer must have

both Windows and OS/2 experience. Depending on the feature, translation code can also be very complex.

9.1.3 General Coding Hints and Tips

There are only a few considerations which programmers need to remember when working with Win32/Open32 programs. The following list summarizes the hints we have after working with Open32.

- Use ULONGs for 32-bit values which need to be accessed by both Win32 and OS/2 code. Often it is impractical to use the actual definition because it exists in only one environment. For example, MPARAM exists only in OS/2 programming.
- If OS/2 code needs to call a Win32 function, create a function in the Win32 source to call it. (See 7.1.2, “How the OS/2 Tree View Control is Written” on page 198)

Appendix A. Common Problems and Easy Solutions

While Open32 certainly makes migrating your Win32 application to OS/2 much easier, many problems can still come up during the process. It is easy to forget one step or forget to modify one line of code. In this section you will find solutions to some of the more common programming missteps and mistakes in working with Open32.

A.1 Compiler Errors

There are only a few compiler errors which might come up in migrating an Open32 application. The most important step, of course, is to replace

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

with

```
#include <os2win.h>
```

in all of your source files. If you forget to do this, the compiler will not recognize any of the Win32 data types or functions.

A.1.1 SYS1041: The name specified is not recognized

This error is a general OS/2 system error which indicates the given program could not be found. When compiling Open32 programs, this may indicate that you are not using the correct compiler name.

A.1.1.1 Problem

```
Operating System/2 Program Maintenance Utility
Version 3.00.008 May  9 1995
Copyright (C) IBM Corporation 1988-1995
Copyright (C) Microsoft Corp. 1988-1991
All rights reserved.

        cl /c howdy.c
        SYS1041: The name specified is not recognized as an
        internal or external command, operable program or batch file.
        NMAKE : fatal error U1077: 'G:\OS2\CMD.EXE' : return code '104
        Stop.
```

A.1.1.2 Solution

You are using the wrong compiler name and need to change its definition in your makefile. VisualAge C++ is executed at the command line, in this case, by:

```
icc /c /Ss howdy.c
```

See the VisualAge C++ documentation for information on command line options.

A.1.2 Errors in Compiling <OS2WIN.H>

If the compiler generates numerous errors when attempting to compile <OS2WIN.H>, you may have forgotten an important command line switch. You must specify the command line option /Ss so that the compiler will accept the double slash (//) as a one-line comment. While the double slash is not defined in standard C, <OS2WIN.H> header file uses it frequently.

A.2 Linker Errors

A.2.1 Obsolete #pragma Warning

When compiling C files which include <os2win.h>, you may get the following warning:

```
H:\TOOLKIT\H\os2win.h(14:4) : warning EDC0523: Obsolete #pragma checkou
ignored. Use #pragma info or the /W options.
```

This message can be safely ignored.

A.2.2 L1104: not valid library

With VisualAge C++ Version 3.0, IBM included a new linker, ILINK. If you get either of the following error messages, you need to change your linker.

A.2.2.1 Problem

```
LINK386 :  fatal error L1104: H:\IBMCPP\LIB\cppos30.lib : not valid libr
```

or

```
LINK :  fatal error L1104: H:\IBMCPP\LIB\cppos30.lib : not valid library
```

A.2.2.2 Solution

You must use the new linker, ILINK, to link your Developer API Extensions application. You cannot use either Microsoft's Segmented-Executable Linker nor the OS/2 Linear Executable Linker from previous toolkits.

A.2.3 Unresolved External on Win32 Functions

During the linkedit you receive unresolved external messages for the Win32 functions you use in your program.

A.2.3.1 Problem

```
howdy.obj(howdy.c) : error LNK2029: "UpdateWindow" : unresolved exte
howdy.obj(howdy.c) : error LNK2029: "RegisterClass" : unresolved ext
howdy.obj(howdy.c) : error LNK2029: "DefWindowProc" : unresolved ext
howdy.obj(howdy.c) : error LNK2029: "GetClientRect" : unresolved ext
howdy.obj(howdy.c) : error LNK2029: "GetStockObject" : unresolved ex
howdy.obj(howdy.c) : error LNK2029: "CreateWindowEx" : unresolved ex
howdy.obj(howdy.c) : error LNK2029: "TranslateMessage" : unresolved
howdy.obj(howdy.c) : error LNK2029: "DispatchMessage" : unresolved e
howdy.obj(howdy.c) : error LNK2029: "PostQuitMessage" : unresolved e
howdy.obj(howdy.c) : error LNK2029: "LoadIcon" : unresolved external
howdy.obj(howdy.c) : error LNK2029: "EndPaint" : unresolved external
howdy.obj(howdy.c) : error LNK2029: "DrawText" : unresolved external
howdy.obj(howdy.c) : error LNK2029: "BeginPaint" : unresolved extern
howdy.obj(howdy.c) : error LNK2029: "MessageBox" : unresolved extern
howdy.obj(howdy.c) : error LNK2029: "LoadCursor" : unresolved extern
howdy.obj(howdy.c) : error LNK2029: "GetMessage" : unresolved extern
howdy.obj(howdy.c) : error LNK2029: "ShowWindow" : unresolved extern
```

A.2.3.2 Solution

You must specify PMWINX.LIB as a library to ILINK so that it can properly connect your Developer API Extensions function calls to the Developer API Extensions dynamic link libraries. Simply add PMWINX.LIB as a parameter to ILINK to resolve this problem.

A.2.4 LNK4021: no stack segment

During the linkedit you receive no stack segment messages.

A.2.4.1 Problem

```
ILink : warning LNK4021: no stack segment
ILink : warning LNK4071: application type not specified; assuming WI
```

A.2.4.2 Solution

You must create a link definition file for your application. The DEF file must at least contain statements naming the application's name, application type, and stack size. See Figure 96 on page 90 for an example.

A.2.5 LNK4038: program has no starting address

During the linkedit you receive the program has no starting address message.

A.2.5.1 Problem

```
ILink : error LNK4038: program has no starting address

There was 1 error detected
```

A.2.5.2 Solution

You must add MAIN.C from \TOOLKIT\SAMPLES\DAPIE\WINMAIN to your application, compile it, and link it in. It contains the start-up procedure main() as required by OS/2. See Section 3.1, “Overview of the Migration Process” on page 79.

A.3 Resource Compiler Errors

The only errors that may come up with resources and the Resource Compiler are caused by not converting the source files to OS/2 format with SMART. This is a very important step because Win32 and OS/2 resource files have completely different formats.

If resource compiler works fine but a graphical resource, such as an icon or a bitmap, does not appear in the application, it is probably because the file has not been converted. Converting graphical resources is a separate step from translating the resource file. The Resource Compiler will not return an error if the icon or bitmap is in Win32 format.

Also note, if a dialog box contains an icon which it cannot load properly, the dialog box may not appear. If you are having problems with dialog boxes that refuse to open, try removing any icons or other resources which may be blocking the dialog's function.

A.3.1 Undefined Keyword or Keyname

A.3.1.1 Problem

The Resource Compiler won't compile converted files. It returns errors such as “undefined keyword or keyname.”

A.3.1.2 Solution

Add the statement:

```
#include <os2.h>
```

to the top of your Resource Compiler file. The Resource Compiler does not natively know the definitions for virtual keys or other constants; it must read them from the standard header files. Note, however, that since resources are not translated at run-time by Developer API Extensions, you cannot include <os2win.h>. The constant definitions differ between OS/2 and Win32, so including <os2win.h> in your resource file may cause unpredictable results. Accelerator keys, for example, will either not work at all, or they will activate the wrong accelerator.

A.4 Run-Time Errors

For the most part, run-time errors do not occur. IBM's high quality development tools detect code that could cause a problem during execution. However, there are a few things which seem innocent enough but which actually cause severe errors.

A.4.1 Program won't load, PMWINX.DLL Access Violation Error

Figure 204 shows the System Error panel which will be displayed when a run-time error occurs for an Open32 program.

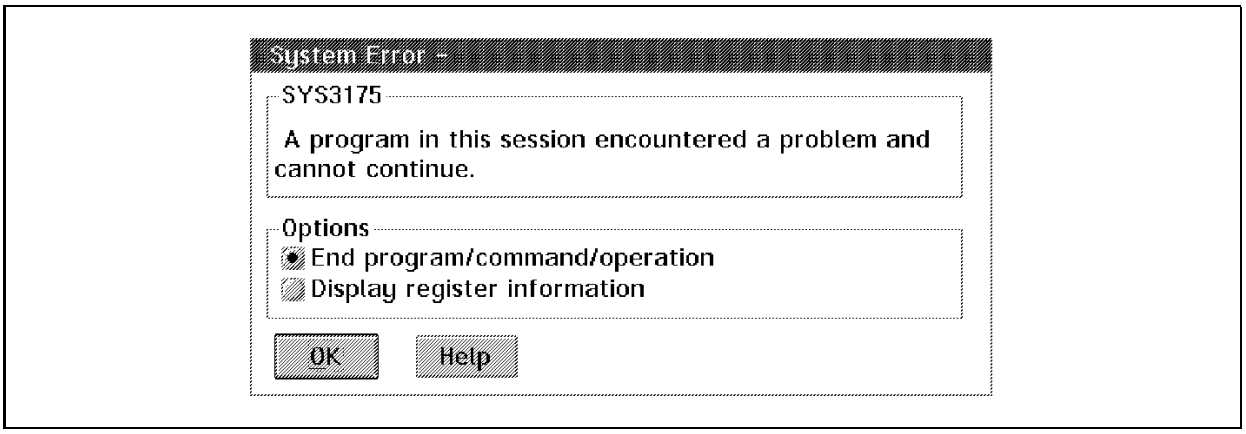


Figure 204. Run-Time Error Message at Program Startup

Figure 205 on page 260 shows the System Error detail panel which can be displayed when the Display register information option is chosen and the **OK** pushbutton is selected from the System Error panel shown in Figure 204.

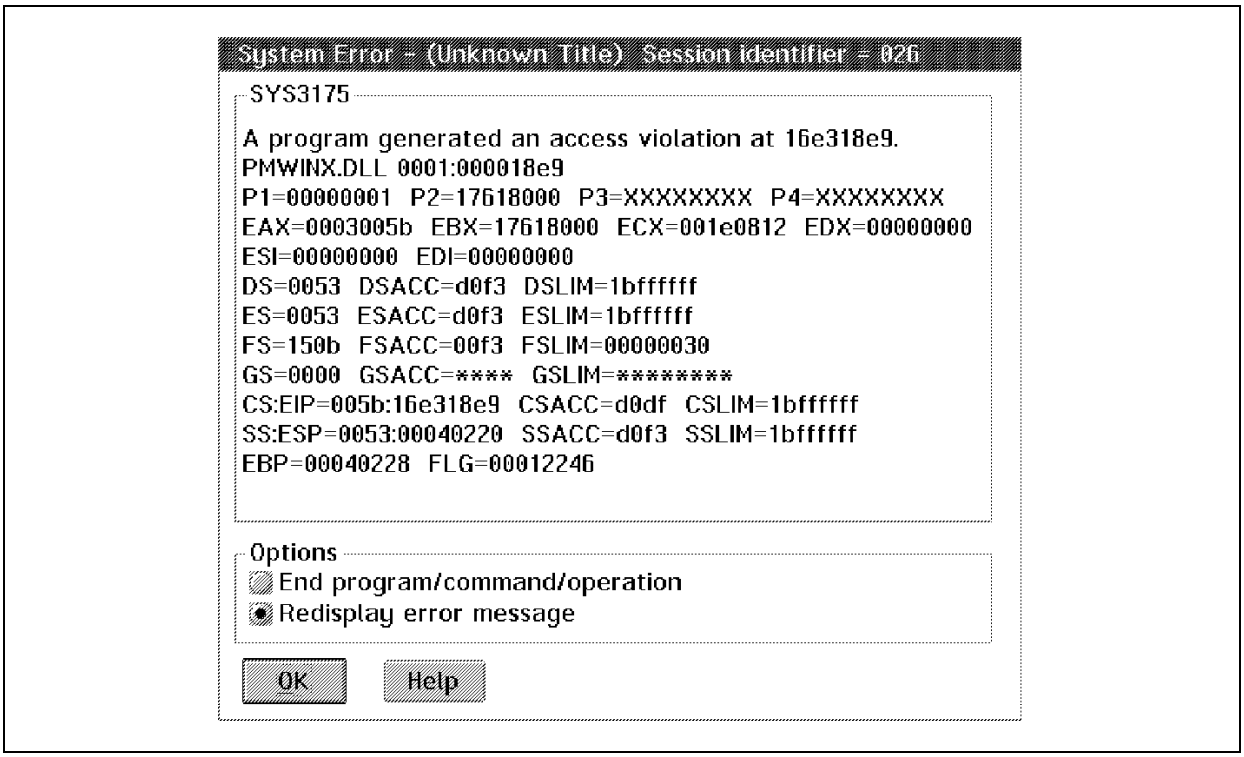


Figure 205. Detailed Information on Run-Time Error

A.4.1.1 Problem

During program startup, OS/2 reports a system error, as shown in Figure 204 on page 259. On the details page (Figure 205), OS/2 shows that PMWINX.DLL encountered a SYS3175 error which is a program access violation in the program PMWINX.DLL.

A.4.1.2 Solution

There are several possible causes of this message. The most likely is that the stacksize of your application is too small. Check your link definitions (DEF) file and make sure the statement:

```
STACKSIZE  65536
```

is in the file. The number following STACKSIZE is the size of the program's stack in bytes. For Open32 applications, the stack must be at least 65536; smaller values may cause run-time errors.

Another probable case of the problem is your resource file. If you changed from string IDs to integer IDs, you may not have caught every instance of a string ID in your program code. A stray call to LoadIcon() or LoadMenu() with a string ID can cause a program crash if the resource does not exist

under the name given. Check your C source code and make sure you use the MAKEINTRESOURCE() macro for all integer IDs.

The problem can also be caused by assigning any resource an ID of 1. When OS/2 attempts to use any such resource (not necessarily when loading it), it will cause a memory violation. Assign the resource another ID.

A.4.2 Dialog Boxes don't Work

A.4.2.1 Problem

If the dialog box contains an icon or a bitmap, it may be causing a problem. Under Win32, a dialog box which contains a resource that cannot be loaded is loaded and executed without the resource. Under OS/2, the dialog box will not load unless all of its component parts also load properly.

A.4.2.2 Solution

Most likely, the ID of the icon in the dialog box does not match a valid icon ID in the resource file. Check to make sure the IDs match.

There is a known problem that a dialog box containing an icon which has a string ID will not load and execute properly under OS/2. You must change the string ID of the icon to an integer.

If the problem continues, try commenting out the definition of the icon. If the dialog box then works, you know for sure that OS/2 is having a problem locating and loading the icon. The icon may not have been converted from Win32 format. If this is the case, use the SMART tool to convert it. See 3.1.4, “Converting Resource Compiler Files” on page 83 for complete details.

A.4.3 Icons or Bitmaps Don't Show

A.4.3.1 Problem

Graphical items such as icons or bitmaps which are bound to the executable using the Resource Compiler do not appear when drawn by the application using DrawIcon() or BitBlt().

A.4.3.2 Solution

This problem may have several different causes. The most likely is that the icon or bitmap has not been converted to OS/2 format. That format differs slightly from the format used in Windows. If you have not used SMART or a similar tool to convert the bitmap or icon to OS/2 format, you need to do so.

If the file is in OS/2 format and problems persist, check to see if the resource ID matches between the Resource Compiler file and your program source code. If you are using string IDs, the name must be in double quotes (“”). If you are using numeric IDs, you must use the MAKEINTRESOURCE() macro.

Appendix B. Special Notices

This publication is intended to help application developers and IBM technical personnel who are interested in building C/C++ applications that utilize Open32 of OS/2 Warp. The information in this publication is not intended as the specification of any programming interfaces that are provided by OS/2 Warp or Win32.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained

elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

BookManager	Hyperwise
IBM	OS/2
Presentation Manager	SOM
System Object Model	VisualAge
Workplace Shell	

The following terms are trademarks of other companies:

C-bus is a trademark of Corollary, Inc.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Microsoft, Windows and the Window 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of SUN Microsystems, Inc.

SMART	One Up Corporation
OpenDoc	Apple Computers, Inc.
Visual C++	Microsoft Corporation
SmallTalk	Digitalk, Inc.
C++	AT&T, Inc.

Other trademarks are trademarks of their respective companies.

Appendix C. Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

C.1 International Technical Support Organization Publications

For information on ordering these ITSO publications see “How To Get ITSO Redbooks” on page 267.

- *VisualAge: Concepts and Features*, GG24-3946
- *Object-Oriented Application Development with VisualAge*, GG24-4227

C.2 Redbooks on CD-ROMs

Redbooks are also available on CD-ROMs. **Order a subscription** and receive updates 2-4 times a year at significant savings.

CD-ROM Title	Subscription Number	Collection Kit Number
System/390 Redbooks Collection	SBOF-7201	SK2T-2177
Networking and Systems Management Redbooks Collection	SBOF-7370	SK2T-6022
Transaction Processing and Data Management Redbook	SBOF-7240	SK2T-8038
AS/400 Redbooks Collection	SBOF-7270	SK2T-2849
RISC System/6000 Redbooks Collection (HTML, BkMgr)	SBOF-7230	SK2T-8040
RISC System/6000 Redbooks Collection (PostScript)	SBOF-7205	SK2T-8041
Application Development Redbooks Collection	SBOF-7290	SK2T-8037
Personal Systems Redbooks Collection	SBOF-7250	SK2T-8042

C.3 Other Publications

These publications are also relevant as further information sources:

- *Designing OS/2 Applications*, ISBN-0-471-58889-X
- *OS/2 Warp Presentation Manager Mentor: Foundations of PM Programming*, ISBN-0-471-13167-9
- *Mastering Windows Utilities Programming with C++*, ISBN-0-7821-1286-2
- *Win32 API Desktop Reference*, ISBN-0-672-30364-7
- *Windows 95 Common Controls & Message API Bible*, ISBN-1-57169-101-7

How To Get ITSO Redbooks

This section explains how both customers and IBM employees can find out about ITSO redbooks, CD-ROMs, workshops, and residencies. A form for ordering books and CD-ROMs is also provided.

This information was current at the time of publication, but is continually subject to change. The latest information may be found at URL `http://www.redbooks.ibm.com`.

How IBM Employees Can Get ITSO Redbooks

Employees may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **PUBORDER** — to order hardcopies in United States
- **GOPHER link to the Internet** - type `GOPHER.WTSCPOK.ITSO.IBM.COM`
- **Tools disks**

  To get LIST3820s of redbooks, type one of the following commands:

  ```
  TOOLS SENDTO EHONE4 TOOLS2 REDPRINT GET SG24xxxx PACKAGE
  TOOLS SENDTO CANVM2 TOOLS REDPRINT GET SG24xxxx PACKAGE (Canadian users only)
  ```

  To get lists of redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET ITSOCAT TXT
  TOOLS SENDTO USDIST MKTTOOLS MKTTOOLS GET LISTSERV PACKAGE
  ```

  To register for information on workshops, residencies, and redbooks:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ITSOREGI 1996
  ```

  For a list of product area specialists in the ITSO:

  ```
  TOOLS SENDTO WTSCPOK TOOLS ZDISK GET ORGCARD PACKAGE
  ```

- **Redbooks Home Page on the World Wide Web**

  `http://w3.itso.ibm.com/redbooks`

- **IBM Direct Publications Catalog on the World Wide Web**

  `http://www.elink.ibmlink.ibm.com/pbl/pbl`

  IBM employees may obtain LIST3820s of redbooks from this page.

- **REDBOOKS category on INEWS**
- **Online** — send orders to: USIB6FPL at IBMMAIL or DKIBMBSH at IBMMAIL
- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank). A category form and detailed instructions will be sent to you.

How Customers Can Get ITSO Redbooks

Customers may request ITSO deliverables (redbooks, BookManager BOOKs, and CD-ROMs) and information about redbooks, workshops, and residencies in the following ways:

- **Online Orders** (Do not send credit card information over the Internet) — send orders to:

	IBMMAIL	**Internet**
In United States:	usib6fpl at ibmmail	usib6fpl@ibmmail.com
In Canada:	caibmbkz at ibmmail	lmannix@vnet.ibm.com
Outside North America:	dkibmbsh at ibmmail	bookshop@dk.ibm.com

- **Telephone orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	(long distance charges apply)
(+45) 4810-1320 - Danish	(+45) 4810-1020 - German
(+45) 4810-1420 - Dutch	(+45) 4810-1620 - Italian
(+45) 4810-1540 - English	(+45) 4810-1270 - Norwegian
(+45) 4810-1670 - Finnish	(+45) 4810-1120 - Spanish
(+45) 4810-1220 - French	(+45) 4810-1170 - Swedish

- **Mail Orders** — send orders to:

IBM Publications	IBM Publications	IBM Direct Services
Publications Customer Support	144-4th Avenue, S.W.	Sortemosevej 21
P.O. Box 29570	Calgary, Alberta T2P 3N5	DK-3450 Allerød
Raleigh, NC 27626-0570	Canada	Denmark
USA		

- **Fax** — send orders to:

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	(+45) 48 14 2207 (long distance charge)

- **1-800-IBM-4FAX (United States)** or **(+1) 415 855 43 29 (Outside USA)** — ask for:

  Index # 4421 Abstracts of new redbooks
  Index # 4422 IBM redbooks
  Index # 4420 Redbooks for last six months

- **Direct Services** - send note to `softwareshop@vnet.ibm.com`

- **On the World Wide Web**

Redbooks Home Page	http://www.redbooks.ibm.com
IBM Direct Publications Catalog	http://www.elink.ibmlink.ibm.com/pbl/pbl

- **Internet Listserver**

  With an Internet E-mail address, anyone can subscribe to an IBM Announcement Listserver. To initiate the service, send an E-mail note to `announce@webster.ibmlink.ibm.com` with the keyword `subscribe` in the body of the note (leave the subject line blank).

IBM Redbook Order Form

Please send me the following:

Title	Order Number	Quantity

O**Please put me on the mailing list for updated versions of the IBM Redbook Catalog.**

First name Last name

Company

Address

City Postal code Country

Telephone number Telefax number VAT number

O Invoice to customer number

O Credit card number

Credit card expiration date Card issued to Signature

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

DO NOT SEND CREDIT CARD INFORMATION OVER THE INTERNET.

List of Abbreviations

DDEML	Dynamic Data Exchange Management Library
DLL	Dynamic Link Library
GDI	Graphical Device Interface
GPI	Graphical Programming Interface
IBM	International Business Machines Corporation
IPF	Information Presentation Facility
ITSO	International Technical Support Organization
MAPI	Messaging Application Programming Interface
MDI	Multiple Document Interface
NMHDR	Notify Message Header
OLE	Object Linking and Embedding
RGB	Red, Green, Blue
RTF	Rich Text Format
SMART	Source Migration Analysis Reporting Tool
SOM	System Object Model
TAPI	Telephone Application Programming Interface
ULON	Unsigned Long Integer
URE	Universal Resource Editor

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

V

W

IBML®

Printed in U.S.A.