Developer's Toolkit for OS/2 2.1

Getting Started

Document Number S60G-9284-00

Note

Before using this information and the product it supports, be sure to read the general information under "Notices" on page ix.

First Edition (April 1993)

The following paragraph does not apply to the United Kingdom or any country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time.

It is possible that this publication may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Requests for technical information about IBM products should be made to your IBM Authorized Dealer or your IBM Marketing Representative.

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 Commercial Relations, IBM Corporation, Purchase, NY 10577.

© Copyright International Business Machines Corporation 1993. All rights reserved.
Note to U.S. Government Users — Documentation related to restricted rights — Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices ix

Preface xi
Who Should Read This Book xi
How This Book Is Organized xi
Developer's Toolkit for OS/2 2.1 xii

Chapter 1. What's New for Version 2.1 1-1
New API Calls, Hooks, and Messages 1-1
Control Program APIs 1-1
Presentation Manager APIs 1-1
Workplace Shell Methods/Classes 1-2
Hooks 1-2
Messages 1-2
New IOCtl Functions 1-3
Lockable Drive IOCtls 1-3
Power Management IOCtls 1-3
CD-ROM IOCtls 1-4
C/C++ Support 1-5
Sample Program Changes and Additions 1-5
PALETTE Sample 1-5
PRINT Sample 1-5
STYLE Sample 1-5
VDD Sample 1-6
WPCAR Sample 1-6
Changes to the Tools 1-6
Information Presentation Facility 1-6
LINK386 1-6
Online Information 1-7
Control Program Guide and Reference 1-7
Debug Kernel Reference 1-7
KwikINF 1-7
Publications Catalog 1-7

Chapter 2. Installing the Toolkit 2-1
System Requirements 2-1
Installation Program 2-1
Installation Procedure 2-2
Subdirectory Structure 2-5
Library Files 2-6

© Copyright IBM Corp. 1993

Header Files 2-6
Include Files 2-7
Installing the Toolkit on Other Operating Systems 2-7

Chapter 3. Online Documents 3-1
Control Program Guide and Reference 3-1
Debug Kernel Reference 3-1
Information Presentation Facility Reference 3-1
Presentation Manager Reference 3-2
REXX Reference 3-2
System Object Model Reference 3-2
Tools Reference 3-3
Using Online Documents 3-3
Finding Help When Using the Enhanced Editor 3-5

Chapter 4. Sample Programs 4-1
Starting a Sample Program 4-1
ANIMALS 4-2
CLIPBRD 4-2
CLOCK 4-3
DIALOG 4-3
DLLAPI 4-3
DRAGDROP 4-4
EAS 4-4
GRAPHIC 4-5
HANOI 4-5
HELLO 4-5
IMAGE 4-5
IPF 4-6
JIGSAW 4-6
NPIPE 4-6
PALETTE 4-7
PDD 4-7
PRINT 4-8
QUEUES 4-8
REXX 4-8
  CALLREXX 4-9
  DEVINFO 4-9
  PMREXX 4-9
  RXMACDLL 4-9
  REXXUTIL 4-10
SEMAPH 4-10
SORT 4-10

STYLE 4-11
TEMPLATE 4-11
TP 4-12
VDD 4-12
VMM 4-13
WORMS 4-13
WPCAR 4-13

Chapter 5. Application-Management Tools 5-1
EXEHDR 5-1
Starting EXEHDR 5-2
FWDSTAMP 5-2
Using Forwarders 5-2
Starting FWDSTAMP 5-3
IMPLIB 5-4
Creating an Import Library 5-4
Starting IMPLIB 5-5
IMPLIB Options 5-5
KwikINF 5-6
Starting KwikINF 5-6
Performing a Search 5-7
LINK386 5-7
Starting LINK386 5-8
Responding to LINK386 Prompts 5-8
Specifying LINK386 Options 5-9
Typing Input on the Command Line 5-9
Creating a Response File 5-10
Example of a Response File 5-10
OS2STUB.EXE 5-11
MARKEXE 5-11
Starting MARKEXE 5-11
Viewing the Application Type 5-12
Setting the Application Type 5-12
MKMSGF 5-13
Creating a Message File 5-13
Starting MKMSGF 5-14
Starting MKMSGF Using a Message Control File 5-14
MSGBIND 5-15
Starting MSGBIND 5-15
Binding the Message File 5-15
NMAKE 5-16
Using NMAKE 5-16
Starting NMAKE 5-17

PACK 5-18
Starting PACK 5-18
Creating a List File 5-20
Starting UNPACK 5-21

Chapter 6. Presentation Manager Tools 6-1
Information Presentation Facility Compiler 6-1
Developing Online Information 6-2
Starting the IPF Compiler 6-3
Compiling Help Files 6-4
Compiling with International Language Considerations 6-4
Viewing an Online Document 6-4
Resource Compiler 6-5
Creating a Resource Script File 6-5
Starting the Resource Compiler 6-6
Dialog Editor 6-6
Starting the Dialog Editor 6-7
Font Editor 6-8
Starting the Font Editor 6-8
Font Resource Files 6-9
Icon Editor 6-9
Starting the Icon Editor 6-10

Chapter 7. SOM and Workplace Shell Tools 7-1
System Object Model Compiler 7-1
Setting the SMINCLUDE Environment Variable 7-1
Setting the SMEMIT Environment Variable 7-1
Setting the SMTMP Environment Variable 7-2
Starting the SOM Compiler 7-2
Running SOM Emitters 7-2
Workplace Class List 7-3
Starting WP Class List 7-4

Chapter 8. System Debug Support 8-1
Communications 8-1
The Debug Files 8-1
Debug Kernel 8-1
Debug Presentation Manager Interface 8-1
Installing the Debug Installation Program 8-2
Installing from Diskette 8-2
Installing from CD-ROM or Remote Disk 8-2
Selecting Installation Options 8-3
Editing the CONFIG.SYS File 8-3

For the Debug Kernel 8-4
Restoring the Kernel 8-4
For the Debug Presentation Manager Interface 8-4
Restoring the Presentation Manager Interface 8-4
MAPSYM 8-5
Starting MAPSYM 8-5
T (Terminal Emulator) 8-5
Hardware Requirements 8-5
Starting T 8-6

Index X-1

Notices

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States and/or other countries:

IBM
MMPM/2
Multimedia Presentation Manager/2
Operating System/2
OS/2
Personal System/2
Presentation Manager
WorkFrame/2
Workplace Shell

The following terms, denoted by a double asterisk (**) in this publication, are trademarks of other companies as follows:

Intel 80386	Intel Corporation
Intel486	Intel Corporation

© Copyright IBM Corp. 1993

Preface

This book describes the individual components that make up the Developer's Toolkit for OS/2 2.1 (Toolkit). Use this book as an overview of the Toolkit, as a guide when installing the Toolkit components, and later as a reference.

Who Should Read This Book

This book is for the professional programmer who will be installing and using the components of the Toolkit. The reader is assumed to be familiar (as a user) with the online services — the tutorial, messages, and helps — provided by OS/2.

How This Book Is Organized

This book contains the following chapters.

Chapter 1, "What's New for Version 2.1," presents a brief summary of the changes to the Developer's Toolkit for OS/2 2.1.

Chapter 2, "Installing the Toolkit," provides a step-by-step procedure of the installation of the Toolkit. A section on library, header, and include files also is included in this chapter.

Chapter 3, "Online Documents," summarizes the contents of the online programming documents and provides a brief description of their features.

Chapter 4, "Sample Programs," explains the OS/2 function every sample program demonstrates.

Chapter 5, "Application-Management Tools," describes the tools that accept 32-bit application code and provides a command-line syntax for starting the tool.

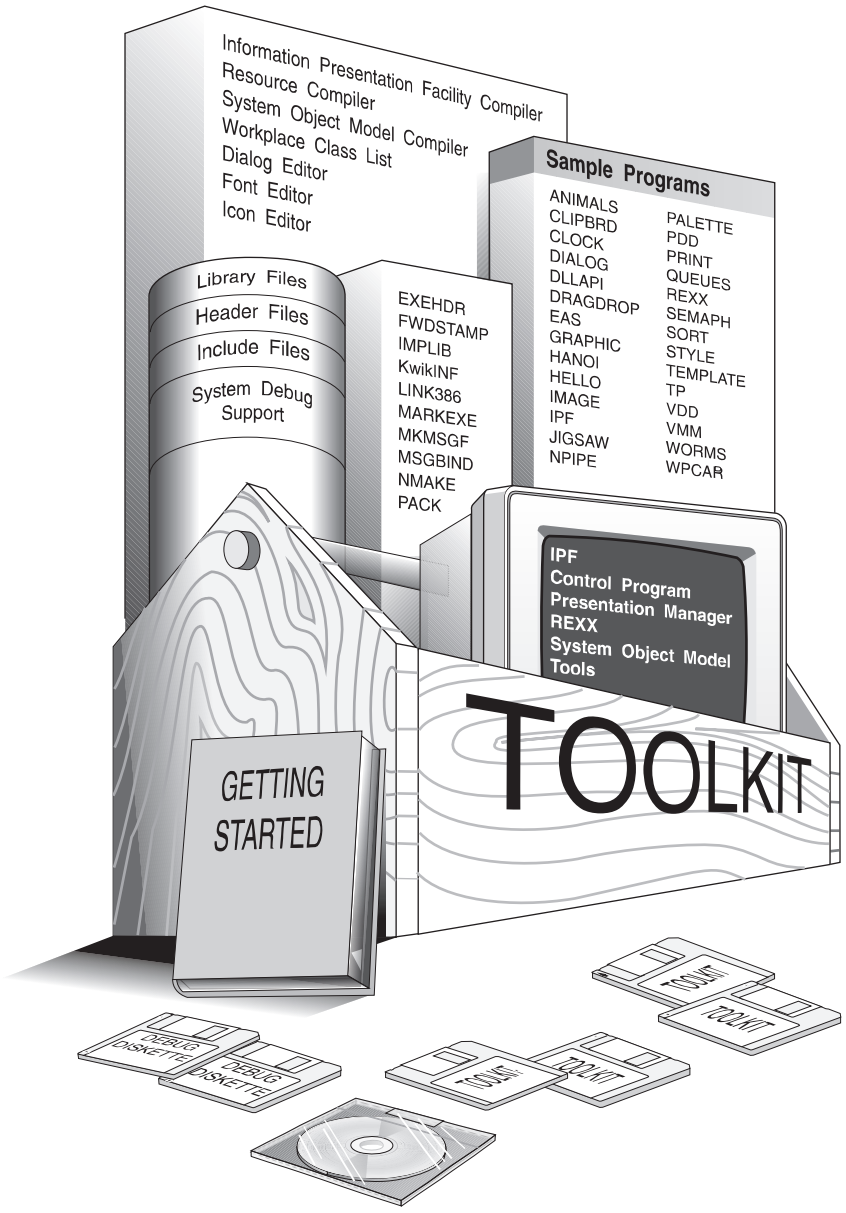
Chapter 6, "Presentation Manager Tools," describes the Resource and Information Presentation Facility compilers and the Presentation Manager resource editors.

Chapter 7, "SOM and Workplace Shell Tools" describes the System Object Model compiler and the Workplace Class List (WP CLass List) program.

Chapter 8, "System Debug Support" describes the interface that installs the debug kernel, and the debug Presentation Manager interface, as well as other tools associated with system debugging.

© Copyright IBM Corp. 1993

Developer's Toolkit for OS/2 2.1



Chapter 1. What's New for Version 2.1

This chapter presents a brief summary of the new features and functions provided in the Developer's Toolkit for OS/2 2.1.

New API Calls, Hooks, and Messages

The following application program interface (API) calls, hooks, and messages are available.

Control Program APIs

These APIs are documented in the online *Control Program Guide and Reference* in the *Toolkit Information* folder:

DosProtectClose
DosProtectEnumAttribute
DosProtectOpen
DosProtectQueryFHState
DosProtectQueryFileInfo
DosProtectRead
DosProtectSetFHState
DosProtectSetFileInfo
DosProtectSetFileLocks
DosProtectSetFilePtr
DosProtectSetFileSize
DosProtectWrite

Presentation Manager APIs

These APIs are documented in the online *Presentation Manager Reference* in the *Toolkit Information* folder:

WinCheckInput
WinLockPointerUpdate
WinLockupSystem
WinQuerySysPointerData
WinSetPointerOwner
WinSetSysPointerData
WinUnlockSystem

© Copyright IBM Corp. 1993

Workplace Shell Methods/Classes

These APIs are documented in the online *Presentation Manager Reference* in the *Toolkit Information* folder:

wpAddPowerPage
wpAddPowerViewPage
wpChangePowerState
wpEjectDisk
wpLockDrive
WPPower
wpQueryAssociatedProgram
wpQueryAutoRefresh
wpQueryDefStatusView
wpQueryDriveLockStatus
wpQueryPowerConfirmation
wpQueryPowerManagement
wpQueryRefreshRate
wpSetAssociatedFileIcon
wpSetAutoRefresh
wpSetDefStatusView
wpSetPowerConfirmation
wpSetPowerManagement
wpSetRefreshRate

Hooks

The following hook is documented in the online *Presentation Manager Reference* in the *Toolkit Information* folder:

HK_LOCKUP

Messages

The following message is documented in the online *Presentation Manager Reference* in the *Toolkit Information* folder:

WM_MOUSEMAP

New IOCtl Functions

New input/output control (IOCtl) functions have been provided for three categories of device management: lockable drives, power management, and CD-ROM. Information on these IOCtls is contained in the *Control Program Guide and Reference* in the *Toolkit Information* folder.

Lockable Drive IOCtls

For lockable drives, the IOCtl functions are as follows:

Category 8, Function 40H — Unlock, Lock, and Eject Media
Category 8, Function 5DH — Diskette Control
Category 8, Function 66H — Query Lock Status

Power Management IOCtls

For power management, the IOCtl functions are as follows:

Category 12, Function 40H — Send Power Event
Category 12, Function 41H — Set Power Event Resource
Category 12, Function 60H — Query Power Status
Category 12, Function 61H — Query Power Event
Category 12, Function 62H — Query Power Info

CD-ROM IOCtls

For CD-ROM, the IOCtl functions are as follows:

Logical Disk IOCtls

Category 08H, Function 63H — Get Device Parameters

Drive and Disk IOCtls

Category 80H, Function 40H — Reset Drive
Category 80H, Function 44H — Eject Disk
Category 80H, Function 46H — Lock/Unlock Door
Category 80H, Function 50H — Seek
Category 80H, Function 60H — Device Status
Category 80H, Function 61H — Identify CD-ROM Driver
Category 80H, Function 63H — Return Sector Size
Category 80H, Function 70H — Location of Drive Head
Category 80H, Function 72H — Read Long
Category 80H, Function 78H — Return Volume Size
Category 80H, Function 79H — Get UPC

Audio IOCtls

Category 81H, Function 40H — Audio Channel Control
Category 81H, Function 50H — Play Audio
Category 81H, Function 51H — Stop Audio
Category 81H, Function 52H — Resume Audio
Category 81H, Function 60H — Return Audio-Channel Information
Category 81H, Function 61H — Return Audio-Disk Information
Category 81H, Function 62H — Return Audio-Track Information
Category 81H, Function 63H — Return Audio-Subchannel Q Information
Category 81H, Function 65H — Return Audio-Status Information

Character Device Driver IOCtls

Category 82H, Function 60H — Return Drive-Letter Information

C/C++ Support

A subset of the OS/2 header files have been C++ enabled and will be installed in the \TOOLKT21\CPLUS\OS2H subdirectory. If you have a C++ compiler installed, you can use these headers to compile C and/or C++ programs. To use these headers, modify the CONFIG.SYS file as follows:

```
SET INCLUDE=x:\TOOLKT21\CPLUS\OS2H;%include%
```

where

x
is the drive where the files are installed

%include%
is the existing INCLUDE= value

If you want to use the OS/2 C++ headers to compile the Toolkit samples, you may need to copy remaining headers that are not C++ enabled from the \TOOLKT21\C\OS2H directory into the \TOOLKT21\CPLUS\OS2H directory.

Sample Program Changes and Additions

This section summarizes the changes and additions to the sample programs.

PALETTE Sample

The PALETTE sample has been added to the Toolkit. PALETTE demonstrates 32-bit graphics functions.

PRINT Sample

The PRINT sample has been expanded to demonstrate selective printing and how to query and display printer forms and margins.

STYLE Sample

The STYLE sample has been changed to show how to determine if a selected font conforms to International Standards Organization 9241 standards.

VDD Sample

The VDD (Virtual Device Driver) sample is now written in C language.

WPCAR Sample

The WPCAR sample has been changed to illustrate exception handling in Workplace Shell objects.

Changes to the Tools

This section describes the tools changes.

Information Presentation Facility

The Information Presentation Facility (IPF) compiler now supports the Brazilian language.

LINK386

Object-oriented compilers may change the name of a function internally. LINK386 now has the ability to change the name back to the original function name when displaying error messages.

Online Information

This section describes the changes and additions to the online information.

Control Program Guide and Reference

The *Control Program Reference* has been replaced. The information in the online *Control Program Reference* has been integrated with the information in the *Programming Guide Volume I*. This new online book is now called *Control Program Guide and Reference* and resides in the *Toolkit Information* folder.

The online *Control Program Guide and Reference* presents information differently than the *Control Program Reference* in the OS/2 2.0 Toolkit. The application program interface (API) calls are presented in an alphabetic list for easy reference. These APIs can also be accessed from the guidance information by clicking on the API name as it appears in the text.

The presentation of the APIs changed as Parameters and Returns no longer appear in the left-hand selection column. To access the parameters, click on the parameter in the syntax panel which appears on the right-hand side of the screen. To see the return codes, click on the **"rc"** field in the syntax panel.

Debug Kernel Reference

The *System Debug Support* information has been removed from the *Tools Reference* information folder. This information has been expanded to include additional debugging commands and placed in its own folder, *Debug Kernel Reference*.

KwikINF

Documentation on the KwikINF utility program has been added to the *Tools Reference* information folder. KwikINF is an online information search and retrieval utility program and is described further in "KwikINF" on page 5-6.

Publications Catalog

Added to the Toolkit directory is a publications catalog, \TOOLKT21\BOOKCAT.ASC. This file is in ASCII format and contains titles and prices of OS/2 documentation and how to order it. You can view this file with any text editor or direct it to a printer for a hardcopy listing.

Chapter 2. Installing the Toolkit

The Toolkit is a collection of sample application programs, programming tools, library files, header files, and online documents designed to aid you in developing applications for OS/2.

System Requirements

This chapter provides a step-by-step procedure that guides you through installing the Toolkit using the program diskettes or CD-ROM. Before we begin, make sure you have the following:

A personal computer that has an Intel386** (or higher) microprocessor with the OS/2 operating system installed
A minimum of 4MB of memory (8MB or more is recommended)
A high capacity 3.5-inch or 5.25-inch diskette drive, or a CD-ROM drive
A hard disk drive with 24MB of free disk space

We recommend that you install a mouse.

Refer to the *Getting Started* book for Multimedia Presentation Manager Toolkit/2 for the system requirements of Multimedia Presentation Manager/2* (MMPM/2)*

Installation Program

The Toolkit installation program is a Presentation Manager* application. Three components are available for installation: "Sample Programs," "Online Information," and "Development Tools." The installation program is preset to install all three components on the drive on which the OS/2 operating system was installed (the default drive). If you want to install the Toolkit on another drive, or install each of the components separately on different drives, you can do so using the installation program menus.

* Trademark of the IBM Corporation

** Trademark of Intel Corporation

© Copyright IBM Corp. 1993

After installation completes, the Toolkit folder on the desktop, illustrated by the toolbox icon, will contain folders for the installed components:

PM Development Tools
Toolkit Information
Sample Programs

You can install the Toolkit for OS/2 2.1 on an OS/2 1.3 system. The installation program will transfer your files and update the environment variables; however, no programs will be registered on the desktop.

Installation Procedure

While you are installing the Toolkit, few decisions are necessary. However, if you need help, you can get it by pressing the F1 key, by selecting the Help push button, or by choosing one of the help choices from the Help menu.

For example, suppose you want information about installing a component on a drive other than the default drive. To access the online help for Toolkit installation:

1. Select **Help** from the menu bar.
2. Select **General help**.

The Toolkit comes on 3.5-inch 1.44MB diskettes, or 5.25-inch 1.2MB diskettes, or CD-ROM. Using the diskettes or CD-ROM as appropriate, follow these steps to install the Toolkit:

Note: If you plan to use the WorkFrame/2* product along with the Toolkit, the recommended installation sequence is:

1. WorkFrame/2
2. Toolkit
3. Compiler

1. Start an **OS/2 Window** or **OS/2 Full Screen** session.
2. Insert Toolkit installation *Diskette 1* in drive A. If installing from a CD-ROM, insert the CD-ROM in the CD-ROM drive.

---

* Trademark of the IBM Corporation

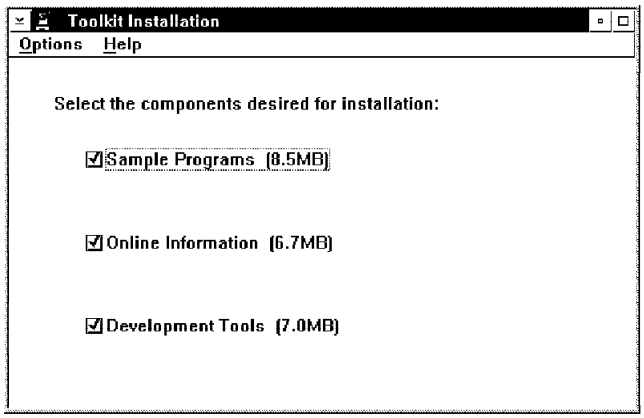
3. Type the following at the command prompt using the appropriate drive letter for the media you are using. The following example shows installation from diskette on drive A:

   a:install

   Note: If you have a system with multiple diskette drives, the Toolkit installation Diskette 1 can be inserted into a diskette drive other than A. For example, you can insert the diskette into drive B and type

   b:install

4. Press Enter and wait for the logo window to appear.
5. Select **OK**. The Toolkit Installation window appears. The window is similar to the following illustration:



6. Mark the box for each component to be installed.
7. Select **Options**.

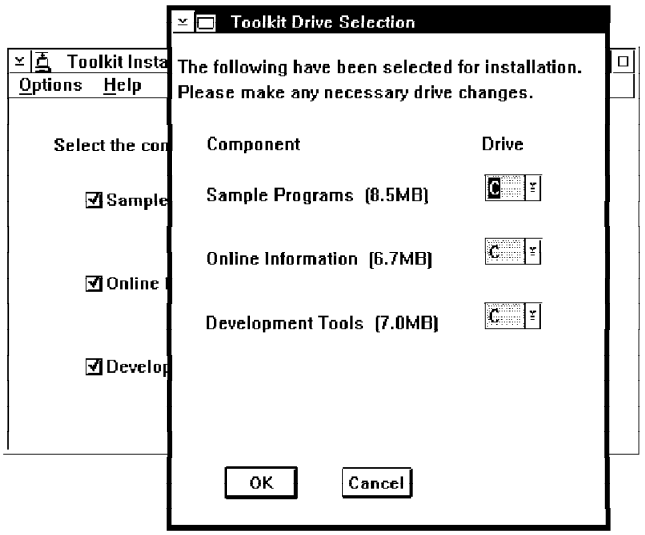
   To install one or more components on the default drive:

   a. Select **Install**.
   b. Follow the prompts in the message boxes to complete the installation on the default drive.

To install components on other drives:

a. Select **Set drives**.

   The Toolkit Drive Selection window shows the selected Toolkit component and the current drive assignment. The window is similar to the following illustration:



   If you did not select a component for installation on the previous window, it will not appear in this window. To install a component on another drive, select the Down arrow in the **Drive** field, then select the desired drive letter. After completing your changes, select **OK** to activate the change. The Toolkit Installation window reappears.

b. Select **Options**.

c. Select **Install**.

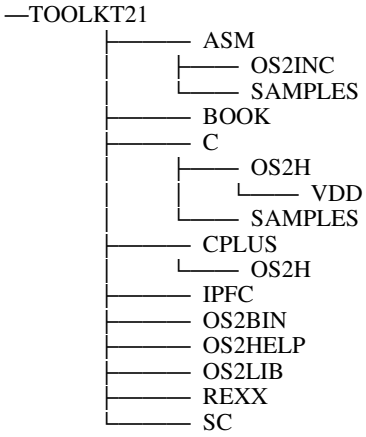
d. Follow the prompts in the message boxes to complete the installation.

If installing from diskette, you will swap diskettes several times during installation. The message Install successful on drive x appears under the name of each component at the completion of the installation. Remove the last diskette or CD-ROM as appropriate. Select **Options**, then **Exit**.

For changes to take effect, you need to shut down your system, then restart it.

Subdirectory Structure

The subdirectory TOOLKT21 is created when you install Toolkit components on a hard disk. The Toolkit can coexist with the *Developer's Toolkit for OS/2 2.0* and *OS/2 Version 1.3 Programming Tools and Information*. The following is the subdirectory structure for the Toolkit:



The subdirectories and their contents are:

ASM\OS2INC	Assembler language include files.
ASM\SAMPLES	Assembler language sample programs.
BOOK	Online documents.
C\OS2H	C language header files.
C\OS2H\VDD	C language header files for virtual device drivers.
C\SAMPLES	C language sample programs.
CPLUS\OS2H	C++ language header files.
IPFC	Information Presentation Facility symbol, language, and tag files.
OS2BIN	Programming tools.
OS2HELP	Help files for Presentation Manager resource editors.
OS2LIB	Library files.
REXX	REXX header files and sample programs.
SC	System Object Model class definition files.

Library Files

Because the OS/2 operating system provides different names and entry points for 16-bit and 32-bit functions, it is possible to mix 16- and 32-bit code within a single .OBJ module. You also can call 32-bit functions from a 16-bit C-language program or call 16-bit APIs from a 32-bit program. To support this mixed-mode programming, the Toolkit provides two sets of library files.

OS2286.LIB The library against which the system calls for 16-bit programs are resolved.

OS2386.LIB The library against which the system calls for 32-bit programs are resolved.

Header Files

Source 32-bit C-language header files (file name extension .H) are provided with the Toolkit. Also provided is a subset of these header files which are C++ enabled. The OS2.H header file includes a set of files. Each of these files contains definitions and macros for the system functions, structures, data types, and symbolic constants associated with a specific group of API functions. The following list provides a brief description of these files:

OS2DEF.H
Defines common constants, types, error codes, and structures.

BSE*.H
Includes all control program (base) API functions.

PM*.H
Includes all Presentation Manager API functions.

REXXSAA.H
Includes all REXX functions.

SOM*.H
System object model functions and definitions.

WP*.H
Workplace object methods.

An include statement at the top of your source-code file automatically calls a hierarchy of header files. You can select the files you want to include by placing statements in your source code **before** the include statement; for example:

```
#define  INCL_
#include  <os2.h>
```

Where INCL_* represents a symbolic identifier (* represents an abbreviation that defines the include file or part of the include file where the API function is declared).

Include Files

Source assembler language include files are provided for the assembler language programmer. OS2.INC is the top-level include file used to call lower-level control program (base) and Presentation Manager include files.

Installing the Toolkit on Other Operating Systems

If you have an operating system other than OS/2, you must use the TKXFER command to copy files to your hard disk. TKXFER will operate on systems that provide DOS emulation. TKXFER is located on diskette 1.

Note: References to "diskette" in this discussion also apply to the installation CD-ROM.

Copy TKXFER to the operating system on which you want to install the Toolkit.

Some of the files on the diskettes are compressed. When the Toolkit is installed under OS/2, the data on the diskettes are *unpacked* automatically. When the Toolkit is installed under any other operating system, the files must be unpacked manually.

Files that are "packed" can be recognized by the @ symbol as the third character in the file-name extension. The OS/2 CD-ROM file system does not support the @ symbol in file names. Other characters should be substituted in the file name. Extensions with only one character have blanks padded with underscores; for example:

Original	**Compressed**
FILE1.COM	FILE1.CO@
FILE2.H	FILE2.H_@

When unpacking manually, do not specify an output file name; TKXFER uses the original uncompressed file name and extension as the destination file name. It also

preserves the date, time, and file attributes of the original uncompressed file from the header of the compressed file.

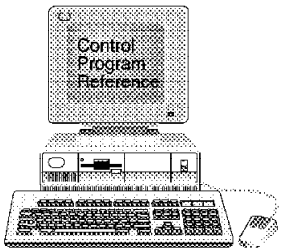
TKXFER copies files that are not compressed and handles file information, such as date, time, and file attributes in the same way the COPY command does. Therefore, a diskette can have both compressed and uncompressed files.

If You Want To:	**Type:**
Copy a compressed file from the current directory on drive A to drive C	TKXFER A:FILE.CO@ C:
Copy an entire diskette of compressed and uncompressed files from the current directory on drive A to the root directory on drive C	TKXFER A:\. C:

Chapter 3. Online Documents

The Toolkit includes the following online documents:

Control Program Guide and Reference
Debug Kernel Reference
Information Presentation Facility Reference
Presentation Manager Reference
REXX Reference
System Object Model Reference
Tools Reference



Control Program Guide and Reference

The *Control Program Guide and Reference* provides the C-language syntax for each of the base operating-system application programming interface (API), including input and output parameters, data structures, data types, return codes, and example code. Guidance information is also provided to assist you in developing applications using these items. API functions (indicated by the prefix "Dos") are presented by component; such as Error Management, Exception Management, and File System. The API functions, within each of the components to which they apply, are listed in alphabetic order. API functions also are available from a single alphabetic list. The online *Control Program Guide and Reference* contains the information in the *Control Program Programming Reference* and the *Programming Guide Volume I*.

Debug Kernel Reference

The *Debug Kernel Reference* provides details on the debug kernel, installation, debugging commands and syntax, and the debug Presentation Manager interface. Additional information on the debug kernel is presented in Chapter 8, "System Debug Support" on page 8-1.

Information Presentation Facility Reference

The *Information Presentation Facility Reference* presents guidance and reference information for the design and development of online documents. It also presents guidance and reference information for a help facility that users of your application can access.

This online reference contains an alphabetic list of Information Presentation Facility (IPF) tags, symbols, and control words. IPF compiler error messages, window

© Copyright IBM Corp. 1993

functions, dynamic data formatting functions, and help manager messages and functions also are included. The online IPF document contains the information in the *Information Presentation Facility Guide and Reference* book.

Presentation Manager Reference

The *Presentation Manager Reference* provides the C-language syntax for all the API functions for the Presentation Manager, including input and output parameters, data structures, data types, return codes, and example code. API function prefixes include Dev (device), Drg (dragdrop), Ddf (dynamic data format), Gpi (graphics), Prf (profile), Spl (spooler), and Win (window). Also included are graphics orders, application hooks, Presentation Manager messages, and the Workplace Shell (WP) methods. The online *Presentation Manager Reference* contains the information in these books plus some additional information:

Presentation Manager Programming Reference, Volume I
Presentation Manager Programming Reference, Volume II
Presentation Manager Programming Reference, Volume III

REXX Reference

The *REXX Reference* provides details of REXX functions, including call syntax, parameters, return values, and error messages. Code examples also are included. The information is presented by component, such as Subcommand Interface, System Exit, Macrospace, Variable Pool Interface, and Halt and Trace. The online *REXX Reference* contains the information in the *Procedures Language 2/REXX Programming Reference* book.

System Object Model Reference

The *System Object Model Reference* is a complete reference for each of the classes and methods used for the object-oriented programming environment. Also included are System Object Model (SOM) C-language and C++ bindings, the Object Interface Definition Language syntax, and the SOM compiler command syntax. The information in this online document contains the *System Object Model Guide and Reference* book.

Tools Reference

The *Tools Reference* presents a collection of the tools that support OS/2 applications development. They are:

EXEHDR
FWDSTAMP
IMPLIB
KwikINF
LINK386
MARKEXE
MKMSGF
MSGBIND
NMAKE
PACK
T

Also included are the Presentation Manager tools:

Dialog Editor
Font Editor
Icon Editor
Resource Compiler
Workplace Class List

Using Online Documents

The online documents in the Toolkit were developed with IPF. IPF displays information through a familiar user interface and lets you do the following:

View a table of contents from which you can quickly gain access to a category
View the category and select related topics from a menu
View multiple windows of related information for comparison values
Search for a topic throughout the document
Copy the contents of a topic to the system clipboard for editing with the OS/2 System Editor, the Enhanced Editor, or any other editor with this capability.
Copy the contents of a topic to a temporary file for editing with a text editor.

When installed, the online documents are added to the *Toolkit Information* folder. To access the online documents, select the folder, then select the appropriate book. A window that has a table of contents (Contents window) will appear. Figure 3-1 shows an example of a Contents window.

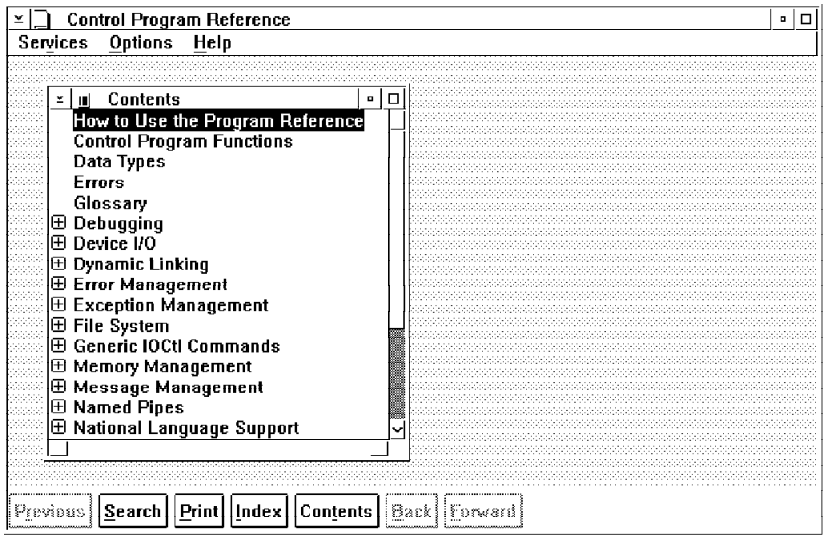


Figure 3-1. Contents Window Example. The Contents window lists available categories.

When the Contents window first appears, some categories have a plus sign (+) beside them. The plus sign indicates that additional topics are available. Click on the plus sign to expand the category.

For more information about using Toolkit online documents, see the "How to Use" sections in each of the documents.

Finding Help When Using the Enhanced Editor

The Enhanced Editor (available with the OS/2 operating system) has a feature that lets you obtain instant access to information in each of the Toolkit online documents. Suppose you invoke the Enhanced Editor, begin writing a window procedure using WinSetFocus, and need information for one of its parameters. To find this information, position the cursor at the beginning of the text string WinSetFocus and press Ctrl + H. The Enhanced Editor takes you directly to the WinSetFocus call in the online *Presentation Manager Reference* document. When you finish using WinSetFocus, toggle back to the Enhanced Editor by pressing Alt + Esc.

Copying the Contents of a Window: You can also copy the contents of WinSetFocus (or any other online topic) for editing with the Enhanced Editor. Here's how:

1. Select **Services** from the menu bar.
2. Select **Copy**. A clock icon appears briefly, indicating that the system has placed a copy of the topic to the clipboard (a temporary holding place).
3. Toggle back (Alt + Esc) to the Enhanced Editor.
4. Move the mouse pointer to where you want the copy to appear. Select **Edit** from the menu bar.
5. Select **Paste**. A copy of the topic is inserted at the entry field of the Enhanced Editor file.

Chapter 4. Sample Programs

This chapter describes the sample programs available with the Toolkit. Most sample programs are written in C language and demonstrate the use of the functions of the control program (base operating system) and the Presentation Manager interface. In addition to C language, there are assembler language and REXX sample programs. Each sample program serves as a template that can be easily modified for your own purposes. All C language samples contain the overhead routines necessary to create a Presentation Manager application, as well as stubs for the basic menu items that all applications should have. There are many comments within the source code that clarify technical information. Names of the sample programs correspond to their Toolkit subdirectory names.

Starting a Sample Program

From the Desktop: When installed, most sample programs appear in the Sample Programs folder. To start a sample program, select the folder, then select the appropriate sample program.

From an OS/2 Command Prompt: To start a sample program from an OS/2 command prompt, type the name of the executable file and press Enter. If you have edited source code of a sample program and want to recompile, link, and run the files; change to the SAMPLES subdirectory and type:

NMAKE /f samples.mak *[samplename]*

where:

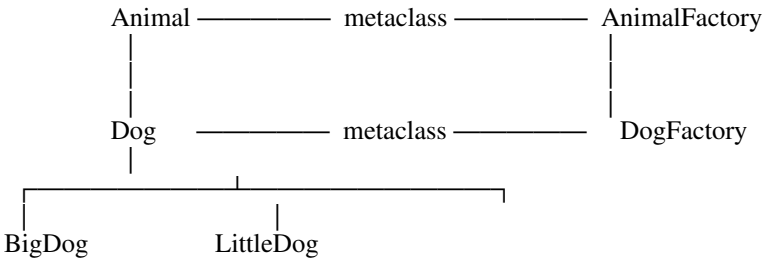
samplename
is the name of the sample program you want to build. To build all of the sample programs, set samplename to *all* (or omit it).

Your output is the executable file. For information about the NMAKE utility, see "NMAKE" on page 5-16.

© Copyright IBM Corp. 1993

ANIMALS

The Basic SOM Sample Program: ANIMALS demonstrates a system object model (SOM) program in an object-oriented programming context. ANIMALS also demonstrates basic concepts of subclassing and inheritance, polymorphism, encapsulation, and constructors using the analogy of a zoological taxonomy. This sample program consists of 6 classes containing 14 methods. The 6 classes and their parentage are shown below:



The *AnimalFactory* and *DogFactory* metaclass supply constructor methods. *BigDog* and *LittleDog* inherit the metaclass of their parent, DogFactory, and their constructors can be used to create BigDogs, LittleDogs, and generic Dogs.

CLIPBRD

The Clipboard Sample: CLIPBRD demonstrates how to provide a Presentation Manager interface to the clipboard. Initially, CLIPBRD displays a standard window with a bit map. The user can cut and paste data in this window, using the system clipboard as an intermediate storage area.

CLOCK

The Timer Services Sample: CLOCK demonstrates how to use and implement window timers and system-resource timers. This sample program displays both an analog and digital clock. To simulate elapsed seconds, the main Presentation Manager thread repeatedly sets a one-second window timer that updates the current time. CLOCK features an audible and visual alarm that the user can set. When the time expires, the sample makes use of the DOS timer services and notifies the user by sounding an alarm.

DIALOG

The Dialog Box Sample: DIALOG demonstrates how to associate a dialog box with a standard window. The dialog box is defined as a dialog template in a resource file. This sample program also demonstrates how to implement an entry field, push buttons, and message boxes.

DLLAPI

The Dynamic Link Library Sample: DLLAPI demonstrates how to write and use a dynamic link library (DLL). The sample has a DLL file and an executable (EXE) file. The DLL uses protected memory on its shared data, and exception management to validate the pointer parameters for a 32-bit API function. The EXE file demonstrates how to handle a divide-by-zero exception, and calls the function with invalid pointer parameters, followed by a call with valid pointer parameters.

DRAGDROP

The Direct Manipulation Sample: DRAGDROP demonstrates how to move files between directories with the dragging techniques of direct manipulation. This sample program creates a list box that contains a scrollable list of the current directory. To change the current directory to one lower in the directory tree, the user selects the directory name from the list and presses Enter. To change the current directory to one higher in the directory tree, the user selects **File** from the menu bar, then **Open.** The **Select subdirectory** window appears. The user types the name of the subdirectory, then selects **OK**.

The sample program must be started twice so there are two running instances of the sample. Then, using a mouse, the user:

Displays a directory file list from the first sample
Selects a file name from the second sample
Drags the file name (with mouse button 2) to the directory in the first sample
Drops the file name in the directory of the first sample.

The file is now moved into the chosen directory.

EAS

The Extended Attributes Editor Sample: EAS demonstrates a multithreaded application that retrieves, modifies, or sorts files by their extended attribute value. Included in this sample program are Presentation Manager procedures for dialog boxes and a standard client window. EAS lets the user select an extended attribute file name from a list, or enter a new name in an entry field. The user can select the extended attribute type from a table.

GRAPHIC

The Graphics Sample: GRAPHIC demonstrates how to use default viewing transformation functions of the Presentation Manager. It also demonstrates how to use an asynchronous drawing thread. The sample program lets the user load metafiles using a dialog box. The dialog box has a help push button. When the help push button is activated, it provides instructions on loading a metafile from another directory. The user also can print a metafile or graphic circle.

HANOI

The Multithreaded Presentation Manager Sample: HANOI demonstrates a multithreaded application with the familiar "towers of Hanoi" puzzle. When the sample program is started the user sees three poles (A, B, and C). Initially, pole *A* has on it a stack of disks starting with the largest disks on the bottom and succeeding smaller disks on the top. The main thread handles the Presentation Manager interface and lets the user start or stop the Hanoi routine. It also lets the user reset the number of working disks. The second thread is created when **Start** is selected from the Options menu. This thread starts the recursive execution of the Hanoi algorithm, runs in the background, and moves and paints the disks.

HELLO

The Standard Window Sample: HELLO demonstrates how to create and display a standard window. This sample program also demonstrates how to use resources defined in a resource script file. Initially, HELLO displays a standard window with the text "Hello." The menu bar contains the **Options** choice. If the choice is selected, the resulting menu contains three choices, each of which paint a different text string in the window.

IMAGE

The Presentation Manager Porting Sample: IMAGE demonstrates how to migrate from an existing OS/2 16-bit application to a 32-bit application. This program also demonstrates how to display an image using the GpiImage function. The image data comes from a file the user selects using the standard File Open dialog procedure.

IPF

The Information Presentation Facility Sample: IPF demonstrates how to use the Information Presentation Facility to create an online document and an application-controlled window that displays animation. The online document also features customized windows that display text and graphics.

Two files are associated with this sample:

The IPF online document (.INF) file
The OS/2 dynamic link library (.DLL) file.

The .INF file is the compiled IPF tag document. The source contains tagging that defines different types of windows. Tags that control the format and display of text also are included in this file.

The .DLL file is the compiled C-language source for the communication object that is called when the .INF file is read during run time. The bit map files used for the animation are also provided.

JIGSAW

The Retained Graphics Sample: JIGSAW demonstrates the use of bit maps in a graphics application. JIGSAW provides a jigsaw puzzle based on the decomposition of an arbitrary bit map loaded from a file. The user can jumble the pieces, then drag them with a mouse. The image can be made smaller or larger, or scrolled horizontally and vertically.

JIGSAW also demonstrates how to call the Information Presentation Facility help hook, to create a help instance and associate the instance with the active application window.

NPIPE

The Named Pipes Sample: NPIPE demonstrates two-way communication between two unrelated processes using named pipe functions. This sample program implements the game of tic-tac-toe with two executable files, CLINPIPE.EXE (the client) and SVRNPIPE.EXE (the server). The server is the computer, and the client is the user.

PALETTE

The Palette Sample: PALETTE demonstrates 32-bit graphics functions including:

Creating a window using a custom palette and animation.
Using menus with switches, and modifying the menu text.
Using multiple threads and semaphores in the Presentation Manager environment.
Displaying graphics on the screen using outline fonts and clip paths.
Online help.

Requirements are an XGA adapter card fully populated with 1MB of RAM and the 32-bit Graphics Engine.

When started, PALETTE displays a standard window with a large OS/2 logo in the foreground. The user has the ability to change the OS/2 logo to the IBM logo. If the user resizes the window, the logo is scaled and redrawn to fit the new window size. The user can also control the animation speed from the PALETTE menu.

The animation is performed by creating a clip path which represents the outline of the logo characters (which are displayed using an outline font); setting the clip path to the presentation space; and then drawing a series of lines to the presentation space. Each line is drawn with an incrementing color index. Palette animation is performed using the 32-bit GpiAnimatePalette function call.

In order for PALETTE to remain responsive to system and user messages, no animation is performed on the main window procedure thread. A second thread is created from which all animation is performed.

PDD

The Physical Device Driver Sample: PDD demonstrates how to construct a physical device driver. Features of this sample include:

Initializing a device driver
Defining a device driver header
Requesting device driver helper services.

Also included are such elements as a strategy routine and interrupt handlers.

PDD is an assembler language sample program. To run PDD, add a DEVICE statement to the bottom of the CONFIG.SYS file. Specify the path and complete file name to PDD. For example:

```
DEVICE=C:\TOOLKT21\ASM\SAMPLES\PDD\PDD.SYS
```

PRINT

The Printer Sample: PRINT demonstrates how to display and print text, graphics, metafiles, and bit maps. It also demonstrates how to:

Query and display printer configurations
Interact with the printer drivers to change job properties
Query and display available printer and screen fonts
Query and display printer forms and setup margins
Selectively print part or all of a document on an asynchronous thread.

QUEUES

The Interprocess Communication Queue Sample: QUEUES demonstrates interprocess communications (IPC) using 32-bit queuing APIs. It consists of two executable programs, SVRQUEUE.EXE and CLIQUEUE.EXE.

SVRQUEUE creates an IPC queue; a named, shared-memory buffer for queue elements; and a shared, named, mutex (mutual exclusive) semaphore. After initializing the queue, SVRQUEUE starts a thread to read from the queue, prints the contents of the messages read from the queue, and terminates at the user's request.

CLIQUEUE opens the queue and accesses the shared-memory element buffer and mutex semaphore, and starts a thread to write to the queue. CLIQUEUE requests a string of data from the user, allocates a shared-memory element from the buffer, puts the string in the shared-memory element, then uses an event semaphore to direct the thread to write the element to the queue. CLIQUEUE terminates at the user's request.

REXX

Five REXX sample programs are included with the Toolkit:

CALLREXX
DEVINFO
PMREXX
RXMACDLL
REXXUTIL

CALLREXX

The REXX Interpreter Invocation Sample: CALLREXX demonstrates how a C-language application calls a REXX application. To run the REXX application BACKWARD.FNC, CALLREXX.C issues RexxStart. RexxStart calls the REXX interpreter and passes it a parameter string. BACKWARD.FNC returns a result string to the C-language application.

DEVINFO

The REXX Variable Pool Interface Sample: This program issues DosDevConfig and returns the data in a collection of compound variables when all available items are requested, or a single variable when only one item is requested. This is a REXX subcommand handler and Variable Pool example. This sample can be run in an OS/2 full-screen session, an OS/2 text-window session, or in PMREXX.

PMREXX

The Presentation Manager REXX Interface Sample: This sample provides a Presentation Manager window in which the user can display the output from a REXX procedure or from any programs called by the REXX procedure. The window has an entry field into which the user can type.

RXMACDLL

The External Functions in REXX Macrospace Sample: This sample demonstrates the macrospace interface with two C-language programs: MACRO.C and RXNLSINF.C, which are compiled into two separate dynamic link libraries.

MACRO.C contains REXX external functions, which perform REXX macrospace operations. RXNLSINF.C contains a REXX external function that provides information related to National Language Support (for example, as a currency symbol and separator). RXMACDLL.CMD uses MACRO.DLL to load NLSMONEY.CMD into the macrospace and calls NLSMONEY.CMD several times to format currency amounts. NLSMONEY.CMD formats the amounts according to the specifications provided by RXNLSINF.DLL.

RXMACDLL can be run in an OS/2 full-screen session, an OS/2 window session, or in PMREXX.

REXXUTIL

The REXX Utility Functions Sample: REXXUTIL demonstrates a set of external functions packaged in a dynamic link library, including:

Use of OS/2 system functions in REXX external functions
Techniques for passing large amounts of data to a REXX program using REXX compound variables as arrays.

REXXUTIL can be run in an OS/2 full-screen session or an OS/2 window session. REXXUTIL cannot be run in PMREXX because some of the functions use video services.

SEMAPH

The Semaphore and Shared Memory Sample: SEMAPH demonstrates the use of mutex and event semaphores. In the sample, several threads share access to the same resource. A mutex semaphore is used to guarantee that only one thread has access to the resource at a time. An event semaphore is used to signal the thread to give up the resource. The event semaphore can be posted by the user or run in auto mode. In auto mode, the event semaphore will be posted at fixed time intervals. A mutex semaphore is used to check for a stop event or for a user signal to give up the resource.

Each thread can display as a different colored square; similarly, the resource can display as a rectangle, the color of which is that of the first thread that owns it.

SORT

The Multithreaded Process Sample: SORT demonstrates the use of multiple threads by performing multiple sorts at the same time. Each sorting algorithm runs from a separate thread. The main thread is used to handle the main window's messages, while the routine that updates the display is run from another thread.

STYLE

The Style-Guide Sample: STYLE demonstrates a Presentation Manager application that conforms with Common User Access* requirements and implements the following controls:

Container
Notebook
Slider
Spin button
Value set.

This sample program also demonstrates secondary windows, such as dialog and message boxes.

STYLE also demonstrates the detection of a font that does not conform to the International Standards Organization (ISO 9241). When the user is running the sample on an ISO-compliant monitor and selects a non-compliant font in the standard font dialog, a message box is displayed to inform the user. The code in STYLE is structured so that the addition of a new function is handled in an efficient manner. For example, to add a new command to an existing menu, you need only add the command to the resource file, then add the appropriate message-processing routines to the STY_USER.C file.

TEMPLATE

The Application Template Sample: TEMPLATE demonstrates the structure common to all Presentation Manager applications. This sample program shows how to structure an application that has more than one source file. TEMPLATE also demonstrates how to:

Create a standard window
Load resources from a resource file
Create a dialog box and a button control
Display a message box
Open and close a file
Print text
Paint a window
Process a message from a menu
Run a thread in the background
Exit a process.

* Trademark of the IBM Corporation

TP

The Advanced SOM Sample Program: TP (text processing) demonstrates an advanced system object model (SOM) program in an object-oriented programming context. TP demonstrates the use of abstract superclasses, and public and private methods. This sample program also demonstrates subclassing, polymorphism, and encapsulation.

TP reads a free-form file of text and markup language and produces a formatted output file. The markup language is deliberately *made up* so that it is clearly defined as a simulation. The input file demonstrates the production of paginated, multi-column text with indentation, page numbers, unordered lists, headers, and footers.

VDD

The Virtual Device Driver Sample: VDD demonstrates a basic virtual device driver, including how to :

Interface with the virtual device driver helper services
Enable and disable system interrupts
Enable and disable a network card interrupt
Simulate interrupts in a DOS session
Initialize the device driver
Define a virtual device header

Use VDD with the physical device driver sample (PDD) to establish interdevice driver communications between a 16-bit physical device driver and a 32-bit virtual device driver.

VDD also demonstrates how to control a network card. This concept can be used to control other peripheral devices.

To run VDD, add a DEVICE statement to the bottom of the CONFIG.SYS file. Specify the path and complete file name to VDD. For example:

```
DEVICE=C:\TOOLKT21\C\SAMPLES\VDD\VDFT.SYS
```

This sample is dependent upon the physical device driver (PDD) sample. The VDD device statement must occur after the PDD device statement in CONFIG.SYS.

VMM

The Virtual-Memory Management Sample: VMM demonstrates the use of virtual memory by using new memory-management functions to allocate and set the attributes of memory. Users can read or write data into memory and reset the attributes using a dialog box. The memory manager protects or opens the virtual memory to read or write operations according to the different attributes of each memory block. To free memory, the user enters the address of the memory.

WORMS

The Mixed-Mode Sample: WORMS demonstrates how to call video (Vio), keyboard (Kbd), and mouse (Mou) 16-bit function from a 32-bit code segment. This sample program displays earth worms aimlessly moving about the screen. Each worm is a separate thread with a unique color combination and movement pattern. When one worm encounters another worm, the color attribute of the first worm is set to red. The user can add or delete worms using the keyboard or mouse.

WPCAR

The Workplace Object Sample: WPCAR demonstrates how to create a workplace object using basic object-oriented programming techniques and the IBM System Object Model (SOM), including:

Initializing an object
Adding settings pages to an object
Saving and restoring the state of an object
Modifying object context menus (adding and deleting menu items)
Querying of object class data
Processing context menu items
Implementing settings page dialog processing.

Chapter 5. Application-Management Tools

The Toolkit provides the following tools:

EXEHDR
FWDSTAMP
IMPLIB
KwikINF
LINK386
MARKEXE
MKMSGF
MSGBIND
NMAKE
PACK

This chapter provides a brief description of each tool with just enough detail to get you started at the OS/2 command line. For complete information about the tools described here, please refer to the online *Tools Reference*.

EXEHDR

EXEHDR provides a listing of the contents of the executable-file header; it also provides a listing of the attributes of all segments in the file.

Uses of EXEHDR include:

Determining whether a file is an application or a dynamic link library
Modifying and viewing the attributes set by the module definition file
Viewing the number and size of code and data segments.

© Copyright IBM Corp. 1993

Starting EXEHDR

You can start EXEHDR and specify all input from the command line. An example of the syntax follows:

EXEHDR *[options] filename*

where:

options
is the name of the EXEHDR option that modifies the file header. Regardless of *options*, EXEHDR always generates a listing of the file header. See the online *Tools Reference* for a description of EXEHDR options.

filename
is the name of the application or dynamic link library file. You can specify any number of files.

FWDSTAMP

FWDSTAMP adds entry points, called *forwarders*, to a dynamic link library file (.DLL). Forwarders point to API functions or other exported code or data. They contain an import reference so that the final target address of the forwarded entry is contained in a different module. A forwarder might be called an *imported export*.

When a file has a fix-up to a forwarded entry point, the loader resolves that fix-up to the address of the entry point that the forwarder imports by traversing the chain of forwarders until the end of the chain (a nonforwarded export) is reached. All forwarders are implicitly exported.

The imported entry point that a forwarder refers to may itself be another forwarder. The loader will process a chain of forwarders until a nonforwarder entry point is encountered.

There is no *run-time* cost to forwarders; however, there is a slight *load-time* cost as the loader resolves forwarder chains with their final addresses.

Using Forwarders

You use forwarders to combine several DLLs into one without having to relink old applications. For example, if MOUCALLS and VIOCALLS were combined into a single DLL called NEWLIB.DLL, then MOUCALLS and VIOCALLS could be replaced with special DLLs containing forwarders to NEWLIB.DLL.

Starting FWDSTAMP

You can start FWDSTAMP and specify all input from the command line. An example of the syntax follows:

FWDSTAMP *infile deffile outfile*

where:

infile
Specifies the name of the dynamic link library file that LINK386 created. Use the file-name extension of DLL.

deffile
Specifies the name of the module definition file (.DEF) that contains the forwarders.

outfile
Specifies the name of the .DLL file that will contain the added forwarders.

Forwarders are specified in the module definition file so that an exported name, which is also imported, is a forwarder. For example:

```
IMPORTS
     VIOMODEWAIT=NEWLIB.123
EXPORTS
      VIOMODEWAIT @ 25
```

In the example, a forwarder entry point for VIOMODEWAIT is created and contains an import reference to NEWLIB.123.

IMPLIB

IMPLIB is used to generate an *import library* (.LIB) file. IMPLIB takes a module definition file (.DEF) as input. For each export definition in the .DEF file, IMPLIB generates a corresponding import definition.

The .LIB file generated by IMPLIB is used as input to LINK386, which creates an executable (.EXE) file. The .LIB file provides LINK386 with information about imported dynamic link functions.

Creating an Import Library

Import libraries are created by IMPLIB and are used to link dynamic link libraries with applications.

Import libraries are similar in some respects to standard libraries:

- You specify import libraries and standard libraries in the same command-line field of LINK386.
- Both types of libraries resolve external references at link time.

However, import libraries differ from standard libraries in that they contain no executable code. Rather, they identify the dynamic link libraries where the executable code can be found at run time.

Creating import libraries is an extra step. Nevertheless, import libraries are recommended for use with all dynamic link libraries for two reasons:

- IMPLIB automates much of the program creation process for you. To use IMPLIB, you need to supply the .DEF file you already created for the dynamic link library. Without an import library, you must create a second .DEF file that explicitly defines all needed functions in the dynamic link library.
- Import libraries make it easier for one person to write a library and another to write the application. Much of the linking process (linking the .DLL file and creating the import library) can be done by the author of the dynamic link library. The import library and associated .DLL file can then be given as a unit to the person linking the application — that person need not worry about creating a .DEF file.

Starting IMPLIB

You can start IMPLIB and specify all input from the command line. An example of the syntax follows:

IMPLIB *[options] implibname {deffile... | dllfile...}*

where:

options
is the name of the option that modifies the output of IMPLIB. All options are described in "IMPLIB Options."

implibname
is the name of the import library created.

deffile
is one or more module definition files that export routines in the dynamic link library.

dllfile
is one or more dynamic link libraries with exported entry points.

Note: You can specify any number of either module definition files or dynamic link libraries.

The following command creates the import library, MYLIB.LIB, from the module definition file, MYLIB.DEF.

```
IMPLIB MYLIB.LIB MYLIB.DEF
```

IMPLIB Options

Syntax	*Description*
/HELP	Displays a short summary of IMPLIB syntax.
/IGNORECASE	Turns case sensitivity off. This is the default.
/NOIGNORECASE	Turns case sensitivity on.
/NOLOGO	Suppresses the copyright screen when IMPLIB starts.

KwikINF

KwikINF provides you with a quick and convenient method of accessing information in online documents stored in the OS/2 BOOKSHELF from anywhere on the desktop, with the exception of DOS or WIN-OS/2 sessions. When KwikINF is started, you can search for a text string of your choice directly or through a pop up window by pressing a user-selectable hot key. Until you configure KwikINF, your KwikINF hot key is ALT+Q.

Starting KwikINF

KwikINF is installed as a program object in the *Toolkit Information* folder. You start KwikINF by double-clicking on the KwikINF program object or by entering KwikINF from the command line of an OS/2 window. You can also start KwikINF automatically when you start OS/2 by placing a shadow of the KwikINF object in the Startup folder in the OS/2 System folder on the desktop. You shadow an object by pressing CTRL + SHIFT while dragging the object to the place where you wish to drop the shadow.

Note: You should not add KwikINF to your STARTUP.CMD if it is followed by an EXIT statement.

You can start, terminate, and configure KwikINF from a command line in an OS/2 window by entering:

KwikINF [no options] [/C] [/T] [/?]

where:

no options
starts KwikINF. After entering this command, the default KwikINF hot key (ALT + Q) is enabled.

/C
opens the Configure KwikINF window. Use this window to select another KwikINF hot key, to select a default online document from the BOOKSHELF to search, and to select the activation behavior of the KwikINF window. Refer to the online *Tools Reference* for detailed information on how to configure KwikINF.

/T
terminates KwikINF and disables the KwikINF hot key.

/?
displays the following information.

```
Usage: KwikINF [Option]
 Option      Description
   /C        Configure  KwikINF
   /T        Terminate  KwikINF
   /?        This short help list
```

Performing a Search

How you initiate a search for information in online documents is dependent on where you are on the desktop when you press the KwikINF hot key:

From an OS/2 full-screen session, a PM VIO or AVIO window, or PM MLE: position the cursor on the string you want to search for and press the KwikINF hot key. KwikINF retrieves the word at the cursor. If you have configured KwikINF to display the KwikINF window when the KwikINF hot key is pressed, KwikINF automatically places the retrieved word in the **Search String** entry field of the KwikINF window. When you press the **Search** push button or Enter, KwikINF displays the information. If you have configured KwikINF to bypass the KwikINF window when the KwikINF hot key is pressed, KwikINF automatically displays the information.

From a graphic-text PM window: press the KwikINF hot key, then type the string you want to search for in the **Search String** entry field of the KwikINF window. The KwikINF text-retrieval feature is not available from graphic-text PM windows.

LINK386

LINK386 is used to translate object files and standard library files into a single executable file. LINK386 also generates dynamic link libraries and device drivers.

LINK386 uses the following files as input:

One or more object files that are linked with any optional library files to form the executable file. Object files usually have a .OBJ extension.

One or more library files. The library files contain object modules that are linked to the object files to form the executable file. Library files usually have a .LIB extension.

A module definition file. The module definition file provides information to LINK386 about the executable file or dynamic link library file it is creating. The module definition file usually has a .DEF extension.

LINK386 produces three types of output files:

An executable file that runs under OS/2 whenever you specify a module definition file that has a NAME statement. The executable file usually has a .EXE extension.

A dynamic link library file. A dynamic link library is produced whenever you specify a module definition file that has a LIBRARY statement. A dynamic link library file usually has a .DLL extension.

A device driver file. A virtual or physical device driver is produced whenever you specify a module definition file that has the VIRTUAL DEVICE or PHYSICAL DEVICE statements. A device driver file usually has a .DRV extension.

Starting LINK386

To link the object files and optional library files of your application, supply input to LINK386 by:

Responding to a series of LINK386 prompts
Typing commands directly at the command prompt
Creating a response file and entering the file name on the command line.

Responding to LINK386 Prompts

To start LINK386, type the following at the command prompt:

```
LINK386
```

Press Enter; a series of prompts appear, one at a time:

```
Object modules [.OBJ]:
Run file [basename.EXE]:
List file [NUL.MAP]:
Libraries [.LIB]:
Definitions file [NUL.DEF]:
```

You can respond using any combination of uppercase and lowercase letters. Enter your responses by pressing Enter.

To extend input to a new line, type a plus sign as the last character on the current line. When the same prompt appears on a new line, you can continue. Do not split a file name across lines.

To select the default response to a prompt, press Enter. The next prompt appears.

To select the default response to the current prompt and all remaining prompts, type a semicolon and press Enter. Note that you must enter the name of at least one object file.

Responses within a command line are separated by commas.

LINK386 supplies the following default file extensions: .OBJ, .EXE, .MAP, .LIB, and .DEF. You can override these extensions by typing the file extension of your choice.

Specifying LINK386 Options

You can specify options anywhere on the response line, except before a comma at the end of a line of characters. If you want to specify more than one option, either group them at the end of a response, or specify them at the end of several responses. Each option must begin with a forward slash (/). For a complete list of options and their descriptions, see the online *Tools Reference*.

To end the linking process at any point, press Ctrl+Break.

Typing Input on the Command Line

You can start LINK386 and specify all input from the command line. An example of the LINK386 command is:

LINK386 *[options] objfiles[,exefile,mapfile,libraries,deffile]*

where:

options
is the name of the LINK386 option. Any number of options may be specified.

objfiles
is the name of the object files that you want linked.

exefile
is the output file that LINK386 created. LINK386 produces either an executable file, a dynamic link library, or a device driver. If you do not specify a file name, LINK386 uses the name of the first object file. Use the file-name extension .EXE if it is an executable file, .DLL if it is a dynamic link library, and .DRV if it is a device driver.

mapfile
is the name of the file that contains the map listing. The default file name extension is .MAP. Use the /M option to include public symbols in this file. (For information on public symbols, see the online *Tools Reference*.) Enter NUL if you do not want a map file.

libraries
is a list of libraries for LINK386 to search. These libraries include standard or import libraries, but not dynamic link libraries. The library names should be separated by plus signs (+) or blank spaces.

deffile
is the name of the module definition file for the executable file or dynamic link library.

Creating a Response File

To operate LINK386 using a response file, you must first create a file that contains the responses you want LINK386 to process. You can give the file any name, and create it with any text editor.

Type the following command at the command prompt:

LINK386 @*filename[.ext]*

The @ symbol tells LINK386 that *filename* is a response file. If the file is not in the working directory, you must specify the path. Begin using a response file at any point on the LINK386 command line or at any LINK386 prompt. The file should contain responses in the same order as the LINK386 prompts. Each response needs to be on a separate line. If you choose to place responses on the same line, separate them with commas.

If the file does not contain responses for all the prompts, LINK386 displays the appropriate prompt and waits for you to supply a response. End the response file with a semicolon.

You can use special characters in the response file the same way you would use them in responses entered at the keyboard.

Example of a Response File

The response file in the following example instructs LINK386 to generate an executable file called FUN.EXE, and four object modules, FUN, SUN, RUN, and GAMES.

```
fun+sun+run+games /map
fun.exe
funlist
;
```

If you specify the file name, FUNLIST, LINK386 will generate a map file named FUNLIST.MAP. Adding the /MAP option will cause LINK386 to include the public symbols of the application in the map file.

OS2STUB.EXE

OS2STUB.EXE is included in the executable file created by LINK386, if the STUB statement is included in the module definition file. The stub is invoked whenever the file is executed under DOS. By default, LINK386 adds its own standard stub for this purpose.

MARKEXE

MARKEXE lets you view and set the type of application. The type of application identifies the OS/2 sessions in which a program can run. You can use MARKEXE in conjunction with programs that you have created using LINK386 or with programs created by some other means.

Starting MARKEXE

The MARKEXE command has the following form:

MARKEXE [force] [no] [display|dllinit|dllterm|*type*|lfns] *filename*

where:

force
Marks the executable file with OS/2 as the target operating system even though the file was marked for another operating system. Using *force* may produce internally inconsistent executable files.

no
Sets the command to the opposite condition.

display
Displays the application type in a message. Does not change the file.

dllinit
Sets per process initialization for the dynamic link library.

dllterm
Sets per process termination for the dynamic link library.

type
specifies the application type of the executable file. It can be one of the following:

WINDOWAPI Uses the API function provided by the Presentation Manager. It must be executed in a Presentation Manager window.

WINDOWCOMPAT Runs (compatible) in a Presentation Manager window or in a full-screen session.

NOTWINDOWCOMPAT The application must execute in a full-screen session only.

UNSPECIFIED The application type is unknown. By default, OS/2 runs an unspecified application type in a full-screen session.

If *type* is not specified, MARKEXE simply displays the current type of the executable file.

lfns
specifies that the program supports long file names.

filename
specifies the executable file to be marked. Any number of files can be marked.

MARKEXE does not modify the file if the application type of the executable file is the same as the requested type. It displays the message unchanged to indicate this.

Viewing the Application Type

You can view the application type of MYPROG.EXE by typing the following:

```
MARKEXE MYPROG.EXE
```

MARKEXE displays the type in a message that looks like this:

```
MYPROG.EXE: OS/2 2.1, WINDOWCOMPACT, LFNS
```

Setting the Application Type

You can set the application type for MYPROG.EXE to WINDOWCOMPAT by typing:

```
MARKEXE windowcompat myprog.exe
```

If you have more than one executable file to be set to the same application type, you can supply the file names in a single command line, as in the following example:

```
MARKEXE windowcompat myprog.exe abc.exe xyz.exe
```

MKMSGF

MKMSGF converts a text message file to an output (binary) message file that DosGetMessage uses to display messages. Text messages in OS/2 full-screen applications do not need to be loaded into memory with the application; they can reside on disk until needed.

You can use the output message file by specifying a message file name and a message number in the DosGetMessage parameter list. The messages also can be bound to the executable file by MSGBIND (see "MSGBIND" on page 5-15).

Creating a Message File

The input message file is a standard ASCII file that contains three types of lines:

Comment
Component identifier
Component message.

Comment lines are the first lines of a file and must begin with a semicolon. A component-identifier is a three-character name identifier (for example, "DOS") that precedes all MKMSGF message numbers. Component-message lines consist of a message header and an ASCII text message.

The following is an example of a text message source file.

```
;This is an example
;of a text message
;file
DOS
DOS 1  E: File not found
DOS 1 1?:
DOS 1 2H: Usage: del [drive:][path] filename
DOS 1 3?:
DOS 1 4I: 1% files copied
DOS 1 5?:
DOS 1 6W: Warning! All data will be erased!
DOS 1 7?:
DOS 1 8?:
DOS 1 9P: Do you wish to apply these patches (Y or N)? %
```

where:

DOS0100E — DOS0109P
identifies message numbers in sequence. The first three characters indicate the component identifier; the four-digit number indicates the message number,

which is followed by a letter (described below), then a colon and blank space. If a message number is not used, type the number, end it with a question mark (?), and leave an empty entry.

E, H, I, P, W
indicates the type of message. Categories include error (E), help (H), information (I), prompt (P), and warning (W).

%0
displays a prompt for input from the user, after which a carriage return and line feed are inserted.

Starting MKMSGF

To start MKMSGF, type:

MKMSGF *infile outfile [option]*

where:

infile
specifies the input file that contains message profiles.

outfile
names the outfile using the three-character component identifier and the .MSG file extension; for example, MES.MSG.

option
Specifies the name of the option that modifies the output file.

For a complete list of options and their description, see the online *Tools Reference*.

Starting MKMSGF Using a Message Control File

A message control file is used to create multiple code page message files. An example of the command-line syntax follows:

MKMSGF @*controlfile*

where:

@controlfile
is the name of the file that contains the control statements used to generate a multiple code page message file.

The @ symbol is not part of the file name; it is a required delimiter.

An example of a message control file follows:

```
root.in root.out /Pcodepage
/Ddbcsrang/ctryid /LlangID,VerId /V
sub.  1 sub1.out /Pcodepage
/Ddbcsrang/ctryid /LlangID,VerId
⋮
sub.  n subn.out /Pcodepage
/Ddbcsrang/ctryid /LlangID,VerId
```

MSGBIND

When the DosGetMessage function is issued, it searches for the message in the message segment bound to the application's executable file, and then the application's message file on a a hard disk. To ensure that a message is displayed quickly, you can bind it to the application's executable file by using the MSGBIND utility program. For each executable file, MSGBIND specifies which message files to scan; for each message file, it specifies which message to include in the executable file.

Starting MSGBIND

To start MSGBIND, type:

MSGBIND [*infile*]

where:

infile
specifies the input file that contains the executable files, output message files, and message numbers that are to be bound.

Binding the Message File

The input file contains three types of lines:

> Executable file
< Message file
Message numbers.

An example of an input file follows:

```
>PROG1.EXE
<\MESSAGES\PRGMSG.MSG
PRG 1
PRG 1 1
PRG 1 2
<\MESSAGES\APP.MSG
APP  1
APP  2
APP  3
```

Where:

>PROG1.EXE
: is the executable file to be modified

<
: defines the first message of a series to be bound, delimited either by the end of the series or a less-than symbol (**<**).

<\MESSAGES\PRGMSG.MSG and <\MESSAGES\APP.MSG
: names the files containing the binary versions of the messages (created by MKMSGF) and their identifying numbers: the three-character component identifier and the four-digit message number.

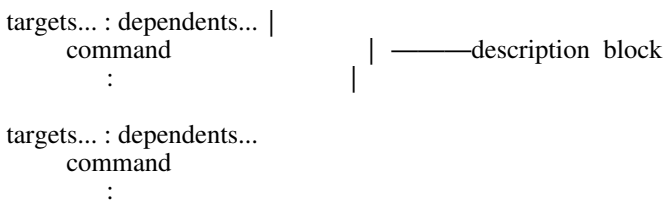
NMAKE

NMAKE carries out all tasks needed to create or update a program after one or more of the source files in the program have changed. NMAKE compares the modification dates for one set of files — the target files — with those of another set of files — the source files. NMAKE then carries out a given task only if a target file is out of date. NMAKE does not compile and link all files just because one file was updated. This can save time when creating programs that have many source files or that take several steps to complete.

Using NMAKE

To use NMAKE, create a description file (or makefile). A description file, in its simplest form, lists which files depend on others and which commands need to be executed if a file changes. You can create an NMAKE description file with any text editor that produces ASCII files.

A description file looks like this:



A dependent relationship among files is defined in a *description block*. A description block indicates the relationship among various parts of the program. It contains commands to bring all components up-to-date. The description file can contain any number of description blocks.

Use NMAKE description files for creating backup files, configuring data files, and running programs when data files are modified.

Starting NMAKE

You can start NMAKE and specify all input from the command line. An example of the syntax follows:

NMAKE *[options][macrodefinitions] [targets][/*F *filename]*

where:

options
is the name of the option that modifies the action of NMAKE. For information about NMAKE options, refer to the online *Tools Reference*.

macrodefinitions
is the name of the macro that replaces one text string for another in the description file. For a list of predefined macros to use with NMAKE, see the *Tools Reference*.

targets
is the name of one or more target files you want NMAKE to create. If you do not list any targets, NMAKE creates the first target in the description file.

/F filename
is the name of the option that specifies *filename* as the name of the description file to use. If a dash (—) is entered instead of a file name, NMAKE reads a description file from the standard input device.

PACK

PACK reduces the size of a file by compressing its data. You can use PACK for a single file or a group of files, thereby reducing the disk space required for your OS/2 application. UNPACK restores a compressed file to its original size and copies it to a specified directory.

Starting PACK

You can start PACK with a single command from the command line. The input required can be specified in one of two ways:

You can type the names of all the files you want to compress directly in the command line (method 1).
You can type the name of a single file that contains a list of all the files you want to compress (method 2).

When using PACK, select the method that is suitable for you. Include the drive and path if the files are not in the working directory. You can specify file names with any combination of uppercase and lowercase letters. File-name extensions are not required; however, if you specify a file name that has an extension, also type the extension.

Examples of the command-line syntax follow:

Method 1:

PACK *sourcefile [packedfile]*
[/H:headerpath\|/H:headerfile|/H:headerpath\headerfile]
[/D:headerdate] [/T:headertime] [/C] [/A] [/R]

Method 2:

PACK *listfile [packedfile] /L*
[/H:headerpath\|/H:headerfile|/H:headerpath\headerfile]
[/D:headerdate] [/T:headertime] [/C]

where:

sourcefile

Specifies the name of the file you want packed (compressed). This parameter is required. Include the drive and path if the file is not in the working directory. Global file-name characters are permitted.

When the data is compressed, the name of the source file is placed in the header of the compressed file and is used as the destination file name during unpacking.

listfile
Specifies the name of the file that contains a list of files that are to be compressed. When naming a list file, do not use global file-name characters.

For information about list files, see "Creating a List File" on page 5-20.

packedfile
Specifies the name of the file that will contain the compressed data. Files that contain compressed data can be recognized by the @ symbol as the last character in the file name. If you do not specify this parameter, PACK places the compressed data in *sourcefile* and modifies its name to contain the @ symbol.

/H:headerpath\ or /H:headerfile or /H:headerpath\headerfile
These parameters can be used separately or paired.

/H:headerpath
Specifies the destination path (drive letters are not permitted) to be placed in the header of the file that contains the compressed data. Unless this path is overridden with the UNPACK command, it will be the destination path when the file is uncompressed. *Headerpath* must end with a backslash (\).

/H:headerfile
Specifies the name of the file to be placed in the header of the compressed file. This file name will be used as the destination file for the uncompressed data and cannot be overridden.

If a header file name is not specified, PACK automatically uses *sourcefile* as the name of the file that is placed in the header of the compressed file.

/H:headerpath\ headerfile
Specifies that both a destination path and a destination file name are to be placed in the header of the file that has the compressed data.

/D:headerdate
Records the date in the header of the file that has the compressed data, and also in the destination file when it is uncompressed.

The date must follow the format /D:MM-DD-YYYY. For example: /D:08-20-1991 and /D:12-30-2010.

/T:headertime
Records the time in the header of the file that has the compressed data, and also in the destination file when it is uncompressed.

The time must follow the format /T:HH.MM. For example /T:02.06 and /T:14.54. Hour 00 represents 12 a.m. and hour 12 represents 12 p.m.

/A

Adds data from *sourcefile* to the data in *packedfile*.

The source file can be either in a compressed or uncompressed state. If the source file is in an uncompressed state, the data is compressed before being added to the file containing the compressed data.

/C

Specifies that the current path be placed in the header of the file that contains the compressed data. When the UNPACK command is used, this path will be the destination path for the file that contains the uncompressed data.

You cannot use /C when the *headerpath* is used.

/L

Indicates that *listfile* is a list file. A list file is not compressed; it simply contains a listing of the names of the files that are to be compressed.

/R

Removes the file specified by *sourcefile* from the file that contains only compressed data. The *sourcefile* parameter must specify the path and file name exactly as they appear in the header of the file with the compressed data; otherwise, the following error message

The specified file to remove was not found.

appears on the screen.

The /R parameter is valid only when used in conjunction with *sourcefile* and *packedfile*.

Note: The path and file-name information stored in the header of the file that contains the compressed data can be displayed by using the /SHOW option available with UNPACK. For information about the /SHOW option, see the UNPACK command in the online *OS/2 Command Reference*.

Creating a List File

To use a list file with PACK, you must first create a file that contains the names of the files you want to compress. You can give the list file any name. Following is an example of specifying a list file at the command line:

```
PACK DEVICE.LST DEVICE.DRV /L
```

The /L indicates that DEVICE.LST is a list file. If the list file is not in the working directory, you must specify the drive and path. Global file-name characters are not permitted in the list-file name. DEVICE.DRV is the destination file for the end-to-end-compressed data. (End-to-end compressed data is the data from each of

the files contained in the list file. This data is stored in a contiguous format in the destination file.)

The syntax used in the list file is similar to that used in the command line. The syntax for a single line in the list file follows:

sourcefile [/H:headerpath\|/H:headerfile|/H:headerpath\ headerfile]
[/D:headerdate] [/T:headertime] [/C]

Remember, when using the list-file method (method 2), global file-name characters are not permitted in the source-file name. Notice also that "PACK" is excluded, and *packedfile* is not permitted in the list file, because they were specified on the command line. You can include comments or blank lines by entering a semicolon as the first character of the line. An example of a list file follows:

```
;This is a comment
C:\OS2\COMMAND.COM
CONFIG.SYS /H:CONFIG.BAK /C
\OS2\INSTALL\DDINSTAL.EXE /H:\OS2\DDINSTAL.TMP /D:1 -15-91 /T:11.45
```

Starting UNPACK

UNPACK restores a file of compressed data to its original size and copies it to a specified drive and path. To start the UNPACK command, type:

UNPACK *sourcefile [destinationdrive:] [destinationpath]*
[/SHOW] [/N:singlefile] [/V] [/F]

where:

sourcefile
Specifies the name of an existing file that contains compressed data. If this file contains one or more files of compressed data, UNPACK restores each file within the file.

destinationdrive:
Specifies the name of the drive to which you want UNPACK to copy one or more restored files.

When you specify a destination drive, but not a path, UNPACK uses the path information stored in the header of the file that contains the compressed data.

destinationpath
Specifies the name of the directory (and its subdirectories) to which you want UNPACK to copy one or more restored files.

When specified, the destination path overrides the path information stored in the header of the file that contains the compressed data.

/SHOW

Displays the destination path and file-name information that are saved in the header of each file containing compressed data.

/N:singlefile

Extracts and uncompresses one file from a file that contains multiple files of compressed data.

/V

Verifies that sectors written to the target disk are recorded properly. This parameter lets you know that critical data has been correctly recorded.

This parameter causes UNPACK to run slower because a check is made for each entry recorded on the disk.

/F

Specifies that files with *extended attributes* should not be unpacked or copied if the destination file system does not support extended attributes.

Chapter 6. Presentation Manager Tools

This chapter describes the Presentation Manager tools:

Information Presentation Facility Compiler — Develop a user-help interface or online documents
Resource Compiler — Add resources to your applications, such as message strings, menus, and dialog boxes
Dialog Editor — Create dialog boxes or changes controls in existing dialog boxes
Font Editor — Modify raster fonts to construct images, such as lines, circles or other geometric shapes
Icon Editor — Create icons, pointers, and bit maps

Information Presentation Facility Compiler

The Information Presentation Facility (IPF) is a set of tools used to create an online help facility for an application. IPF also is used to create online information that can be viewed independent of an application. It is a tool for both the information author and the application programmer.

As an author of online information, you can define the windows in which information is displayed. For example, a window can be split so that scrollable text can be displayed beside a stationary illustration that the text describes. Figure 6-1 shows an IPF split-window design that describes the IBM* Personal System/2* Model 90 XP 486 series.**

* Trademark of the IBM Corporation

** Trademark of Intel Corporation

© Copyright IBM Corp. 1993

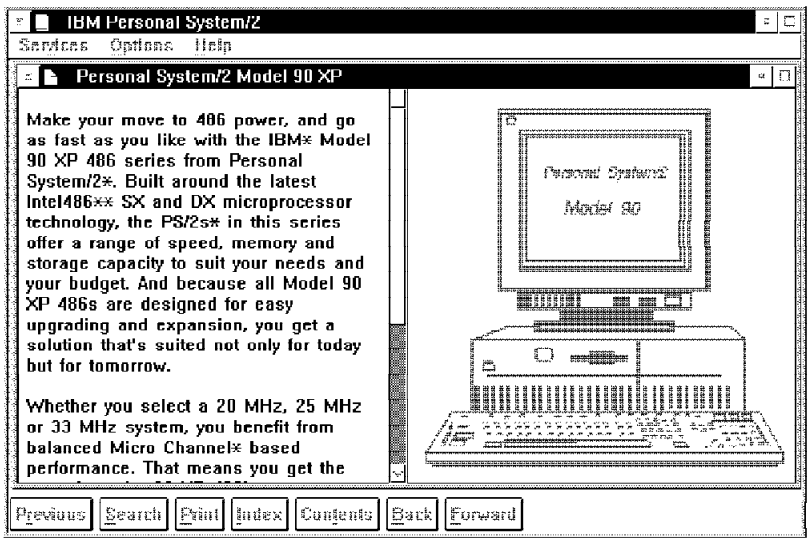


Figure 6-1. IPF Split Window. The split window was created with the IPF tagging language.

Developing Online Information

IPF makes it possible for you to design your information in two types of formats:

An online document format for reference books.
A help-window format for context-sensitive help for your application programs.

To produce either format, you must create a text file using a text editor and two IPF elements:

The IPF tagging language — it consists of the instructions for formatting and displaying your document on the screen.

The IPF compiler — it interprets the tags and converts the source file into an IPF format.

Starting the IPF Compiler

You can start the IPF compiler and specify all input from the command line. An example of the syntax follows:

IPFC *filename* [/INF] [/S] [/X] [/W*n*] [> *messageoutputfilename*]

where:

filename
Specifies the name of your IPF source file.

If you do not give a file-name extension, the IPF compiler uses .IPF by default. If your file has a file-name extension other than IPF, include that file-name extension in the command line.

/INF
Compiles the source file as an online document.

If this parameter is not included, the default is to compile the source file as a help library, whose extension is .HLP.

/S
Suppresses the performance of the Search function. This parameter increases compression of compiled data by about 10% to further reduce the storage it requires.

/X
Generates and displays a cross-reference list.

/W*n*
Generates and displays a list of error messages. The *n* indicates the level of error messages you want to receive. Values you can specify for *n* are 1, 2, or 3.

Warning Level 1 (the most severe)
Warning Level 2 (moderately severe)
Warning Level 3 (the least severe and is the default).

messageoutputfilename
Specifies the name of the file where error and cross reference messages are sent. If you do not specify this parameter, messages generated by **/X** and **/W***n* are sent to the display screen.

Compiling Help Files: To compile a source file that is intended as a help-text window, use the IPFC command without the /INF option. For example:

```
IPFC myhelp.hlp
```

Compiling with International Language Considerations: The following parameters provide international language support:

/COUNTRY=*nnn* (*nnn* is the 3-digit country code)

/CODEPAGE=*nnn* (*nnn* is the 3-digit code page)

/LANGUAGE=*xxx* (*xxx* is a 3-letter identifier that indicates an international languages file is to be used).

An example of the command-line syntax follows:

IPFC *myfile.txt /INF /COUNTRY= 33 /CODEPAGE=437 /LANGUAGE=FRA*

Viewing an Online Document: If you want to see your formatted online document, you can use the VIEW command to display it.

An online document has an extension of INF. It can be viewed by entering its name as a parameter to the VIEW command; for example:

```
VIEW myfile
```

You do not need to include the INF file extension.

Note: You cannot use VIEW to display help-text windows for application programs. However, for test viewing purposes, you can compile the help text as INF files and use VIEW to look at the information.

For complete information about this tool, see the online *Information Presentation Facility Reference*.

Resource Compiler

The OS/2 Resource Compiler is a tool that lets you add application resources, such as message strings, pointers, menus, and dialog boxes, to your application's executable file. The primary purpose of the Resource Compiler is to prepare data for applications that use functions such as WinLoadString, WinLoadPointer, WinLoadMenu, and WinLoadDlg. These functions load resources from the application's executable file or another specified executable file. The application then can use the loaded resources as needed.

The Resource Compiler and the resource functions let you define and modify application resources without recompiling the application itself. The Resource Compiler can modify the resources in an executable file at any time without affecting the rest of the file. You can create custom applications from a single executable file by using the Resource Compiler to add the custom resources you need to each application. The Resource Compiler is especially important for international language support because it lets you define all language-dependent data, such as message strings, as resources. Preparing the application for a new language is simply a matter of adding new resources to the existing executable file.

Creating a Resource Script File

All resources are defined in a resource script file. You use a text editor to create a resource script file that has a .RC extension. Resources are defined either explicitly in statements in the resource script file, or in other files (such as output files from the resource editors). The .RC file is the input file to the Resource Compiler; the output has a .RES extension. The .RC file can contain statements that define resources and that include resources from other files. Text-based resources such as menus, accelerator keys, and text strings are defined in the .RC file. Non-text-based resources are specified in the .RC file as file names of the external files where these resources reside. Such resources include icons, pointers, and bit maps. The syntax for including external files in a resource script varies according to the nature of the resources defined or contained in the files. Fonts have a resource file to themselves.

Make sure that none of the include files in your resource script file contain an end-of-file character. When the Resource Compiler sees an end-of-file character, it assumes it to be the end of all input.

For an example of a resource script file, see the sample program “TEMPLATE.”

Starting the Resource Compiler

You can start the resource compiler in three ways:

Compile a resource script file and bind it to an executable file
Compile a resource script file but do not bind it to the executable file
Compile a resource script file and put it in a dynamic link library.

Compiling and Binding Resources to an Executable File: To compile the resource script file EXAMPLE.RC and bind the resulting compiled resource (.RES) file to the executable file, EXAMPLE.EXE, use the following command:

```
RC EXAMPLE
```

You do not need to specify the .RC extension for EXAMPLE. The Resource Compiler program creates the resource file EXAMPLE.RES and then adds the compiled resource to the executable file EXAMPLE.EXE.

Compiling Without Binding Resources to an Executable File: To compile the resource script file, EXAMPLE.RC, into a resource file without binding the resources to an executable file, use the following command:

```
RC -R EXAMPLE
```

The compiler creates the resource file EXAMPLE.RES.

Putting Resources in a Dynamic Link Library: Instead of binding a resource file to your application, you can put it in a dynamic link library. To add the compiled resources to a dynamic link library, use the following command:

```
RC EXAMPLE.RES DYNALINK.DLL
```

You can then link the file at run time and load the resources into your application by using the DosLoadModule or GpiLoadFonts functions. However, you cannot switch from binding resources to putting resources into a dynamic link library without changing your application source code. For information on how to put resources into a dynamic link library, see the *Application Design Guide*. For complete information about the Resource Compiler, see the online *Tools Reference*.

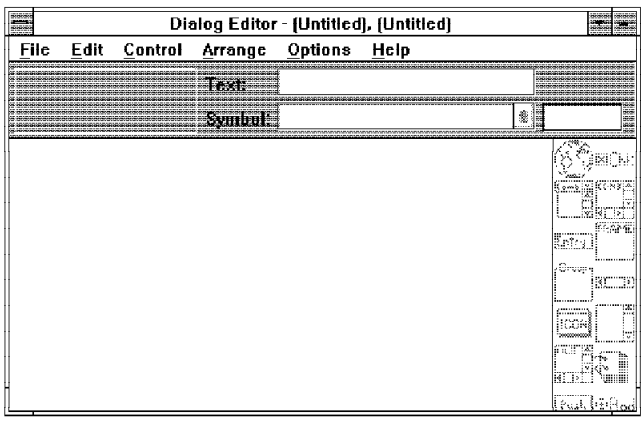
Dialog Editor

The Dialog Editor is used to create and modify dialog boxes and specify the controls and text within dialog boxes. As you create a dialog box and add controls, the Dialog Editor draws it to the screen. You can resize and reposition the dialog box, then test its controls before you incorporate it in your application.

Although the Dialog Editor draws box outlines and controls to the screen so that you can view it from a user's perspective, the Dialog Editor does not save it as a graphic. Instead, the Dialog Editor stores a description of the dialog box and its controls in a text file that has a file-name extension of DLG. It also creates a compiled form of the .DLG file into a resource file that has an extension of RES. The dialog-box and resource files can each contain descriptions of more than one dialog box. The resource file can contain other application resources, such as icons, bit maps, and string tables. It is attached to the application's executable (.EXE) file after the compile and link processes.

Starting the Dialog Editor

To start the Dialog Editor, select the **PM Development Tools** folder, them select **Dialog Editor**. The following window appears.



The **File** and **Edit** menu bar choices provide two ways to create a dialog box:

From the **File** menu, select **New**. This opens new resource files with the extensions .RES and .DLG, but the Dialog Editor does not tell you that it has opened the resource files. You can open a new include file or an existing one.

From the **Edit** menu, select **New Dialog**. The editor opens a new dialog in the current file.

When you edit a dialog box, the names of the resource and include files are shown in the title bar of the Dialog Editor. If you are editing a new file that has not yet been named or saved, *(Untitled)* appears in the title bar in place of a name. If *(Untitled)** appears in the title bar in place of a name, there are unsaved changes.

For more information about the many functions of the Dialog Editor, see the online *Tools Reference*.

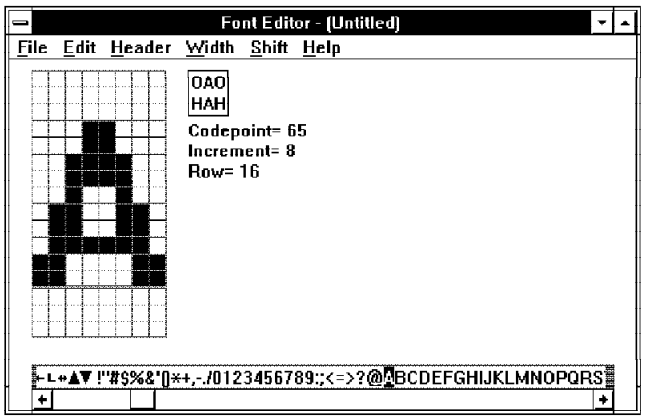
Font Editor

The Font Editor is used to design and save fonts for use in applications. A font is a set of alphanumeric characters, punctuation marks, and other symbols that share a common typeface design and line weight.

When the Font Editor creates a font file, it supplies an FNT file-name extension. The font file contains a header, which describes the font in general terms, and a section that contains bit maps of the characters themselves.

Starting the Font Editor

To start the Font Editor select the **PM Development Tools** folder, then select **Font Editor**. The following window appears.



The quadrille to the left of the screen has within it an enlarged version of the character selected from the long, scrollable, horizontal box at the bottom of the screen. To edit the enlarged version of the character in the quadrille, use the mouse to switch the enlarged representation to black or white. You can change a series of pels by holding mouse button 1 down and moving the pointer through the pels.

Several choices are available from the menu bar that enable you to tailor individual fonts. With these choices you can:

Create a font file or open an existing file
Edit a new or existing font
Define the characteristics of the font
Specify character spacing (fixed or proportional)
Name the typeface

Identify a type style (italic, underscored)
Change the width and weight of individual characters
Insert or delete a column in the character.

Font Resource Files

All resources, except fonts, can be bound to the application's executable file or compiled into a dynamic link library (DLL). Fonts must be put in a separate DLL using the Resource Compiler. You then link the file at run time and load the resources into your application by using the DosLoadModule or GpiLoadFonts function. A DLL containing font resources must have a file-name extension of FON. The .FON file can be installed on the system.

For more information about the Font Editor, see the online *Tools Reference*.

Icon Editor

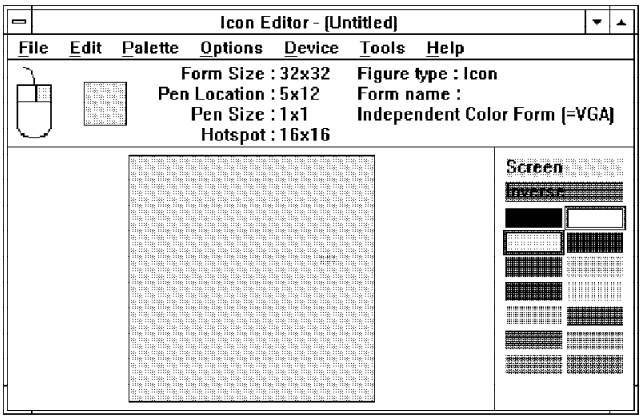
The Icon Editor is used to to create icons, pointers, and bit maps. In the Presentation Manager, an icon is a graphic symbol that identifies a data object, a system action, or a minimized application. A pointer is a small shape on the screen that reflects the movement of the mouse. Pointers have a *hot spot* that identifies their exact location on the screen.

Icons, pointers, and bit maps produced by the Icon Editor are graphic symbols comprised of pels in any of the following display states:

Black
White
Color
Screen (background color)
Inverse screen (inverse of background color).

Starting the Icon Editor

To start the Icon Editor, select the **PM Development Tools** folder, then select **Icon Editor**. The following window appears:



Notice the information area at the top of the Icon Editor window; the items displayed from left to right include:

A two-button mouse, showing the color currently selected for each button
An actual-size image of the current figure that you are editing
A status area that provides:
- Size (in pels using x and y coordinates)
- Pen location
- Pen size (from 1-by-1 to 9-by-9)
- Hot spot (for icons and pointers, but not bit maps)
- Figure type (icon, pointer, or bit map)
- Form name.

The palette window, in the lower-right corner, displays the colors that are available for use during editing. The colors currently selected are marked with frames.

The editing window is the largest part of your working area. Use the mouse to paint the enlarged representation with the selected color.

The menu-bar choices provide access to the many functions of the Icon Editor. These choices enable you to:

- Create a new icon, pointer, or bit map
- Edit an existing icon, pointer, or bit map
- Test the new icon or pointer
- Superimpose a grid over the editing window (for drawing a symmetrical figure)
- Restrict a drawing to straight vertical or horizontal lines
- Make transparent pels (for icons or pointers) visible
- Change the shape and size of the pen
- Select system preferences (to set prompts or suppress warnings)
- Define hot spots (where the mouse pointer is directed).

For more information about the features of the Icon Editor, see the online *Tools Reference*.

Chapter 7. SOM and Workplace Shell Tools

This chapter describes the SOM compiler and the Workplace Shell tool WP Class List. These tools enable you to:

Implement workplace objects

Create workplace object classes and instances of workplace object classes

System Object Model Compiler

The OS/2 operating system provides a programming interface that allows applications to implement desktop objects. This programming interface enables you to create desktop objects that conform to the CUA architecture using basic object-oriented programming techniques. The interface is implemented using the IBM System Object Model (SOM).

Setting the SMINCLUDE Environment Variable

The SOM compiler uses an environment variable called SMINCLUDE to locate included class definitions. Because every SOM class will have an include file for its parent class definition, you must set SMINCLUDE before running the SOM compiler. Its form is similar to the OS/2 PATH or DPATH environment variables, in that it can consist of one or more directory names, separated by a semicolon. Directory names can be specified with absolute or relative path names. For example:

```
SET  SMINCLUDE=.;..\mySCdir;C:\TOOLKT21\SC\INCLUDE;
```

Setting the SMEMIT Environment Variable

SMEMIT is used to indicate which emitter programs should be executed. Like the SMINCLUDE environment variable it can consist of a list of items separated by semicolons. Each item designates a particular emitter by the name of the file extension the emitter produces. For example:

```
SET  SMEMIT=H;IH;PH;SC;DEF;
```

indicates that the EMITH.EXE, EMITIH.EXE, EMITPH.EXE, EMITSC.EXE and EMITDEF.EXE programs should be executed to produce .H, .IH, .PH, .SC, and .DEF files, respectively. By default all emitted output files are placed in the same directory as the input file named in the SC command. If the SMEMIT environment variable is not defined, the SOM Compiler will perform a syntax check of your class definition but no output will be produced.

© Copyright IBM Corp. 1993

Setting the SMTMP Environment Variable

The SMTMP environment variable specifies the name of a directory that the SOM Compiler uses to hold intermediate output files. For example,

```
SET SMTMP=%TMP%
```

tells SC.EXE to use the same directory for temporary files as given by the setting of the TMP environment variable.

As a general rule, the directory indicated by SMTMP should never coincide with the directory used by the SOM Compiler for its input or the emitted output files. If you do not give a setting for the SMTMP environment variable, SC will use the root directory of the current drive for temporary files.

Starting the SOM Compiler

The SOM compiler is actually a precompiler and a collection of code emitters that produce binding files from the output of the precompiler. The files have several forms, including C-header files, a C-implementation template, and the language-neutral version of the class definition file.

To start the SOM precompiler from the command line, type:

SC *[-options] filename [csc]*

where:

options
is the name of the SOM compiler option. Options can be specified individually, as a string of option characters, or as a combination of these forms. Any option that takes an argument must be specified individually or be the final option in a string of option characters.

For a complete list of SOM compiler options, see the online *System Object Model Reference*.

filename
is the name of a file that contains an OIDL class definition. If you do not specify a file-name extension, the compiler uses .CSC by default.

The SOM compiler (SC.EXE) produces .SOF and .SCF files using the file name you specify.

Running SOM Emitters

You complete the SOM compilation process by running the emitters. You can control the output of the emitters from the command line by typing:

COMMAND [-o *filename*] [-a *name[=value]] filestem*

where:

command
is one of the following:

EMITH
EMITPH
EMITIH
EMITC
EMITDEF
EMITSC
EMITPSC
EMITCSC

-o
is an explicit name (including drive, path, and file-name extension) for the emitted output file. If this option is not specified, the output file is placed in the current directory, and the file-name extension defaults to a type appropriate to the selected emitter program.

-a name[=value]
adds a global attribute. Attributes are listed in the online *System Object Model Reference*.

filestem
is the file stem of the .SOF file produced by SC.EXE to use as the basis for emissions.

Workplace Class List

The workplace class list is a tool that creates a workplace object class and an instance of a workplace object class. Workplace objects are constructed using the SOM protocol and are instances of one of the following workplace object classes:

Predefined
These classes are defined by the system. Examples of predefined workplace object classes are WPObject, WPFileSys, and WPAbstract.

Subclass
These classes are derived from existing predefined workplace object classes. They add or remove function; however, they retain the basic behavior of that class.

Replaced

These classes replace the class being *subclassed*. They modify the behavior of an instance of a predefined workplace object class without the instance being aware of the new class.

Starting WP Class List

To start WP Class List, select the **PM Development Tools** folder, then select **WP Class List.** A window similar to the following appears:

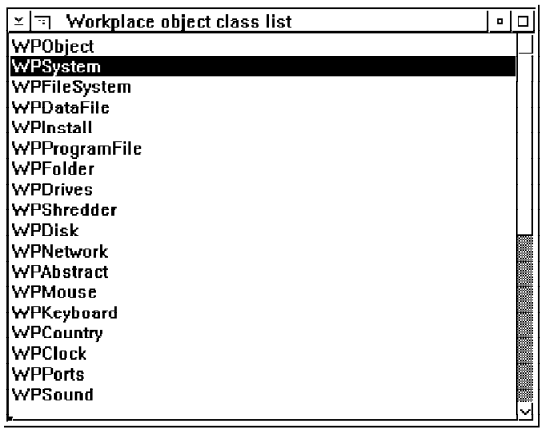


Figure 7-1. The workplace object hierarchy.

Using this window, you can:

Add, delete, and browse registered Workplace Class Objects
Create an instance of a Workplace Class Object
View the registered Workplace Class Object in the system

For more information about this tool, see the online *Tools Reference*.

Chapter 8. System Debug Support

This chapter introduces you to the interface that installs the debug kernel, symbol files, and debug version of the Presentation Manager. It also describes tools that support your debugging efforts. Additional details on the debug kernel can be found in the online *Debug Kernel Reference* in the *Toolkit Information* folder.

Communications

Local and remote debugging are the same, except for the location of the system to be debugged (also known as the *system under test*). If the system to be debugged is close to the debug terminal, use a null modem cable to connect them. If the system is physically distant, use modems. The default setup for the communication port of the debug kernel is:

Baud rate	9600
Parity	none
Data bits	8
Stop bits	1

The Debug Files

The files described in this section are referred to as either a *retail* or *debug* version. "Retail" refers to the files that came with your OS/2 operating system; "debug" refers to the files that came with the Toolkit Debug Diskettes or CD-ROM.

Debug Kernel

The debug kernel, a special version of the OS/2 kernel, makes it possible to set breakpoints and trace programs. It also permits the use of symbolic addresses. You can interact with the debug kernel by using a modem or null modem and a second asynchronous debug terminal.

Note: You can only use the debug kernel files with the version of the OS/2 operating system with which they are associated.

Debug Presentation Manager Interface

The debug Presentation Manager interface is a special version of the Presentation Manager dynamic link libraries. The debugger detects errors in your Presentation Manager application and issues messages to the terminal. This interface is not required to run the debug kernel.

© Copyright IBM Corp. 1993

Installing the Debug Installation Program

The menu-based debug installation program installs debug replacement files for the kernel and the Presentation Manager interface. Once the program is installed, you can install other debug files, or restore retail files, from the OS/2 command prompt.

During initial installation, two files are copied to the root directory of your specified installation drive:

DBINST.CMD	A command file that can be executed separately. This file calls DBUGINST.EXE with the requested installation drive as a command-line argument.
DBUGINST.EXE	This executable file is the user interface. The user can choose which parts of the debug system to install, or which parts to restore to the retail version.

To install from diskette, refer to "Installing from Diskette." Refer to "Installing from CD-ROM or Remote Disk" if you are are installing from CD-ROM or a remote disk.

Installing from Diskette

To install and start the debug installation program:

1. Insert Debug Diskette 1 in your diskette drive.
2. At the OS/2 command prompt, type the following:

   *a:*install *c a*

   where:

   c is the drive letter where the OS/2 operating system is installed.
   a is the diskette drive letter.

   Note: Do not type a colon after the drive letters.
3. Press Enter.

Continue installation by referring to "Selecting Installation Options" on page 8-3.

Installing from CD-ROM or Remote Disk

To install and start the debug installation program:

1. Insert the Toolkit installation CD-ROM into the CD-ROM drive. If installation is from a remote disk, ensure you have access to it.
2. At the OS/2 command prompt, type the following:

d:\OS2TK21\DEBUG\CDINST *d c*

where:

d is the drive letter of the CD-ROM or remote disk.
c is the drive letter where the OS/2 operating system is installed.

Note: Do not type a colon after the drive letters.

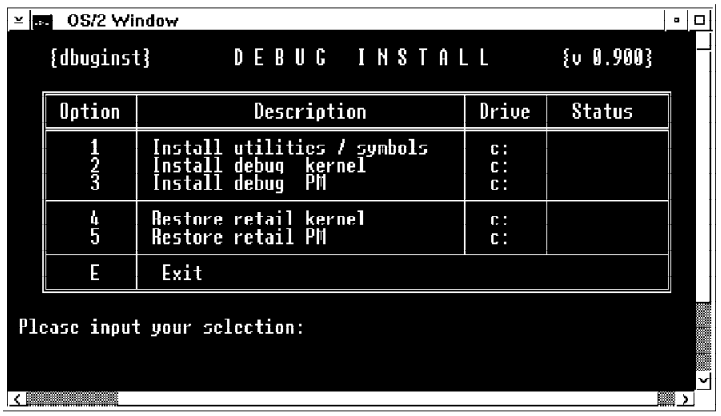
3. Press Enter.

Continue installation by referring to "Selecting Installation Options."

Selecting Installation Options

The user interface consists of a menu that provides installation choices in three optional parts. It also provides the ability to restore two of those parts to their corresponding retail versions.

The following is an illustration of the screen that appears:



When prompted to enter a debug installation option, choose the options in the order they appear on the screen.

Review "Editing the CONFIG.SYS File" after your selections are complete.

Editing the CONFIG.SYS File

When you complete the debug installation procedure, you *may* need to edit your CONFIG.SYS file. The following paragraphs explain:

For the Debug Kernel: If you installed only the debug kernel, shutdown and restart your system.

Restoring the Kernel: To restore the retail kernel, run the debug installation program and select the "**Restore retail kernel**" option.

For the Debug Presentation Manager Interface: If you have installed the debug version of the Presentation Manager interface, modify the DEVICE statement with the PMDD.SYS line as follows:

```
DEVICE=C:\OS2\DEBUG\DLL\PMDD.SYS /Cn
```

The DEVICE statement includes the C drive as the installation drive and allows you to call the debug version of PMDD.SYS from the OS2\DEBUG\DLL subdirectory. The /C switch is set with *n* as the communication port for the debug output.

Modify the LIBPATH statement by adding the DEBUG\DLL subdirectory as follows:

```
LIBPATH=C:\OS2\DEBUG\DLL; ....
```

Shutdown and restart your system to have the changes take effect.

Restoring the Presentation Manager Interface: To restore the retail Presentation Manager, do the following:

1. Restore the device statement:

   ```
   DEVICE=C:\OS2\PMDD.SYS
   ```

2. Modify the LIBPATH statement by removing the DEBUG\DLL subdirectory.
3. Shutdown and restart your system to have the changes take effect.

For a detailed description of the Installation and Restore options available with this interface, see the online *Debug Kernel Reference* in the *Toolkit Information* folder.

MAPSYM

MAPSYM is used to generate binary files that the debug kernel uses to associate a symbolic name with an address in memory.

Starting MAPSYM

MAPSYM creates public symbol (.SYM) files from map (.MAP) files. You must start MAPSYM from the directory in which the map file is located. An example of the syntax follows:

MAPSYM *filename [options]*

where:

filename
is the name of the map file. You do not have to type the .MAP file-name extension.

options
is the name of the MAPSYM option that modifies the action of MAPSYM. For information about MAPSYM options, refer to the online *Tools Reference*.

Note: Be sure the .SYM files are in the same subdirectory as their corresponding DLLs.

T (Terminal Emulator)

T is a terminal emulator and is used by the debug kernel to communicate with the system to be debugged. You can use any ASCII terminal emulator; the Toolkit provides T. A terminal emulator allows a device, such as a personal computer, to enter and receive data from a computer system as if it were a particular type of attached terminal. For example, you use T to send and receive ASCII files.

Hardware Requirements

Make sure your system has a properly installed asynchronous-port and communication-port driver, and that your CONFIG.SYS file has the following line:

```
DEVICE=C:\OS2\COM.SYS
```

Starting T

You can start T at the command line by typing its executable name: T

A blank screen appears. Press the F1 key; a menu appears that lets you:

Display function-key assignments
Set up communication-port parameters
Set the file name and start sending
View the text that has scrolled off the screen
Send the text that was written to a screen, to a file (capture mode)
Toggle to the capture mode
Set the file name or delete the current capture file
Exit from the terminal program.

Note: Capture mode can be started automatically when T is executed by placing the line `Capture=yes` in the initialization file.

For more information about T, see the online *Tools Reference*.

Index

Special Characters

@controlfile 5-14

Numerics

16- to 32-bit sample program 4-5
16-bit library file 2-6
32-bit library file 2-6

A

ANIMALS sample program 4-2
ASM subdirectory 2-5
assembler language
  PDD sample program 4-7
assembler language include files 2-7

B

BOOK subdirectory 2-5
BSE*.H 2-6

C

C subdirectory 2-5
C-Language header files 2-6
CALLREXX sample program 4-9
CLIPBRD sample program 4-2
CLOCK sample program 4-3
Common User Access sample program 4-11
compiler
  Information Presentation Facility 6-1
  Resource Compiler 6-5
  System Object Model 7-1
compiling command for IPF 6-3
control file for MKMSGF 5-14
Control Program Guide and Reference (online document) 3-1
copying the contents of a window 3-5

D

Debug Kernel Reference (online document) 3-1
debug version of Presentation Manager 8-1
debug, kernel 8-1
default file extensions for LINK386 5-9
device driver sample program 4-7
DEVINFO (REXX) sample program 4-9
Dialog Editor 6-6
DIALOG sample program 4-3
DLLAPI sample program 4-3
DRAGDROP sample program 4-4

E

EAS sample program 4-4
Enhanced Editor, using the 3-5
event semaphores sample program 4-10
EXEHDR 5-1
exiting installation 2-4
extended attributes sample program 4-4

F

file extensions for LINK386, default 5-9
Font Editor 6-8
FWDSTAMP 5-2

G

GRAPHIC sample program 4-5

H

HANOI sample program 4-5
hardware requirements 2-1
header files 2-6
HELLO sample program 4-5
help windows, compiling 6-4

© Copyright IBM Corp. 1993

I

Icon Editor 6-9
IMAGE sample program 4-5
IMPLIB 5-4
Information Presentation Facility Compiler 6-1
Information Presentation Facility Reference (online document) 3-1
installation procedure 2-2
international language documents, compiling 6-4
international language support (RXMACDLL) 4-9
IPF command for help files 6-4
IPF sample program 4-6
IPFC command 6-3
IPFC subdirectory 2-5

J

JIGSAW sample program 4-6

K

kernel, debug 3-1
KwikINF 5-6

L

library files 2-6
LINK386 5-7
local debugging, debug kernel 8-1

M

MAPSYM 8-5
MARKEXE 5-11
migration sample program 4-5
MKMSGF 5-13
MSGBIND 5-15
multiple thread PM sample program 4-5
multiple thread sample program 4-10
mutex and event semaphores sample program 4-10

N

named pipe functions sample program 4-6
national language support (RXMACDLL) 4-9
NMAKE 5-16
NPIPE sample program 4-6

O

online document, viewing an 6-4
online documents
- Control Program Guide and Reference 3-1
- Debug Kernel Reference 3-1
- Information Presentation Facility Reference 3-1
- Presentation Manager Reference 3-2
- REXX Reference 3-2
- System Object Model Reference 3-2
- Tools Reference 3-3
- using 3-3

ordering information
OS2.H 2-6
OS2.INC 2-7
OS2286.LIB 2-6
OS2386.LIB 2-6
OS2BIN subdirectory 2-5
OS2DEF.H 2-6
OS2H subdirectory 2-5
OS2HELP subdirectory 2-5
OS2INC subdirectory 2-5
OS2LIB subdirectory 2-5
OS2STUB.EXE 5-11

P

PACK 5-18
PALETTE sample program 4-6
PDD sample program 4-7
physical device driver sample program 4-7
PM*.H 2-6
predefined object class 7-3

Presentation Manager debug version 8-1
Presentation Manager Reference (online document) 3-2
PRINT sample program 4-8

Q

QUEUES sample program 4-8

R

remote debugging, debug kernel 8-1
replaced object class 7-4
Resource Compiler 6-5
retained graphics sample program 4-6
REXX subdirectory 2-5
REXXSAA.H 2-6
REXXUTIL sample program 4-10
RXMACDLL sample program 4-9

S

sample programs
- ANIMALS 4-2
- CALLREXX 4-9
- CLIPBOARD 4-2
- CLOCK 4-3
- DEVINFO 4-9
- DIALOG 4-3
- DLLAPI 4-3
- DRAGDROP 4-4
- EAS 4-4
- GRAPHIC 4-5
- HANOI 4-5
- HELLO 4-5
- IMAGE 4-5
- IPF 4-6
- JIGSAW 4-6
- NPIPE 4-6
- PALETTE 4-6
- PDD 4-7
- PRINT 4-8
- QUEUES 4-8
- REXX 4-8
- REXXUTIL 4-10

sample programs *(continued)*
- RXMACDLL 4-9
- SEMAPH 4-10
- SORT 4-10
- starting 4-1
- STYLE 4-11
- TEMPLATE 4-11
- TP 4-12
- VDD 4-12
- VMM 4-13
- WORMS 4-13
- WPCAR 4-13

SC subdirectory 2-5
SEMAPH sample program 4-10
semaphore sample program 4-10
software requirements 2-1
SOM (advanced) sample program 4-2
SOM (basic) sample program 4-12
SOM (see System Object Model)
SORT sample program 4-10
standard window sample program 4-5
starting the sample program 4-1
STYLE sample program 4-11
subclass of an object class 7-3
symbol files for debug kernel 8-1
System Object Model Compiler
- SMEMIT environment variable 7-1
- SMINCLUDE environment variable 7-1
- SMTMP environment variable 7-2
- starting 7-2

T

TEMPLATE sample program 4-11
terminal emulator 8-5
terminate LINK386 process 5-9
thread sample program, multiple 4-10
TKXFER command 2-7
tools
- Dialog Editor 6-6
- EXEHDR 5-1
- Font Editor 6-8
- FWDSTAMP 5-2
- Icon Editor 6-9
- IMPLIB 5-4

tools *(continued)*
Information Presentation Facility
Compiler 6-1
KwikINF 5-6
LINK386 5-7
MAPSYM 8-5
MARKEXE 5-11
MKMSGF 5-13
MSGBIND 5-15
NMAKE 5-16
PACK 5-18
Resource Compiler 6-5
System Object Model Compiler 7-1
T (terminal emulator) 8-5
WP Class List 7-3
Tools Reference (online document) 3-3
TP sample program 4-12

U

UNPACK 5-18
UNPACK command 5-21
unpacking Toolkit files 2-7

V

VDD sample program 4-12
VIEW command 6-4
viewing an online document 6-4
virtual device driver sample program 4-12
virtual memory sample program 4-13
VMM sample program 4-13

W

workplace class list (tool) 7-3
workplace object sample program 4-13
WORMS sample program 4-13
WPCAR sample program 4-13