IBM

International Technical Support Centers

OS/2 Version 2.0
Volume 3: Presentation Manager and Workplace Shell

GG24-3732-00

OS/2 Version 2.0
Volume 3: Presentation Manager and Workplace Shell

Document Number GG24-3732-00

April 1992

International Technical Support Center
Boca Raton

Take Note!

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

First Edition (April 1992)

This edition applies to OS/2 Version 2.0.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, International Technical Support Center
Dept. 91J, Building 235-2 Internal Zip 4423
901 NW 51st Street
Boca Raton, Florida 33432 USA

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Abstract

This document describes the Presentation Manager component of OS/2 Version 2.0. It forms Volume 3 of a five volume set; the other volumes are:

- *OS/2 Version 2.0 - Volume 1: Control Program*, GG24-3730
- *OS/2 Version 2.0 - Volume 2: DOS and Windows Environment*, GG24-3731
- *OS/2 Version 2.0 - Volume 4: Application Development*, GG24-3774
- *OS/2 Version 2.0 - Volume 5: Print Subsystem*, GG24-3775

The entire set may be ordered as *OS/2 Version 2.0 Technical Compendium*, GBOF-2254.

This document is intended for IBM system engineers, IBM authorized dealers, IBM customers, and others who require a knowledge of Presentation Manager features, functions, and implementation under OS/2 Version 2.0.

This document assumes that the reader is generally familiar with the function provided in previous releases of OS/2.

PS (142 pages)

Acknowledgements

The project leaders and editors for this project were:

Hans J. Goetz
International Technical Support Center, Boca Raton

Giffin Lorimer
International Technical Support Center, Boca Raton

The authors of this document are:

Alan Chambers
IBM United Kingdom

Karsten Düring
IBM Germany

Gert Ehing
IBM Germany

Franco Federico
IBM United Kingdom

Dennis Lock
ISM, South Africa

Joachim Müller
IBM Germany

Jouko Ruuskanen
IBM Finland

Neil Stokes
IBM Australia

Katsutoshi Suzuki
IBM Japan

This publication is the result of a series of residencies conducted at the International Technical Support Center, Boca Raton.

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

Lori Brown
IBM Programming Center, Boca Raton.

Sam Casto and his staff
IBM Programming Center, Boca Raton.

Ann Ford
IBM Programming Center, Boca Raton.

Alfredo Gutiérrez
IBM EMEA Education Center, Boca Raton.

Steve Robinson and his staff
IBM PRGS, Cary.

David Kerr
IBM Programming Center, Boca Raton.

Peter Magid
IBM Programming Center, Boca Raton.

Michael Perks
IBM Programming Center, Boca Raton.

Tom Richards
IBM PRGS, Cary

Thanks also to the many people, both within and outside IBM, who provided suggestions and guidance, and who reviewed this document prior to publication.

Thanks to the following people who created the excellent tools used during the production of this document:

Dave Hock (CUA Draw)
IBM PRGS, Carry.

Jürg von Känel (PM Camera)
IBM Yorktown Heights.

Contents

Abstract iii

Acknowledgements v

Special Notices xv

Preface xvii

Related Publications xix
Prerequisite Publications xix
Additional Publications xix

Chapter 1. Introduction to the Presentation Manager and Workplace Shell . . 1
1.1 The Vision 1
1.1.1 The Workplace Shell User with Older Applications 2
1.1.2 What About the Experienced PC User? 3
1.2 What is Presentation Manager? 3
1.3 What is the Workplace Shell? 4
1.3.1 Workplace Shell as an Operating System Shell 4
1.3.2 Workplace Shell as an Application Environment 5
1.3.3 WPS Objects versus SOM Objects 5
1.4 Presentation Manager Enhancements in OS/2 Version 2.0 5
1.4.1 New Controls and Dialogs 5
1.4.2 Information Presentation Facility Enhancements 6
1.5 Programming Environment 7
1.6 Summary 8

Chapter 2. Presentation Manager Components 9
2.1 Windows 9
2.1.1 Frame Area 11
2.1.2 Client Area 12
2.1.3 Parent and Child Windows 13
2.2 Dialog Boxes 13
2.3 Message Boxes 14
2.4 Control Windows 15
2.5 Icons 16
2.6 Clipboard 17
2.6.1 Shared Memory and the Clipboard 19
2.7 Summary 19

Chapter 3. New Presentation Manager Features 21
3.1 New Window Classes 21
3.1.1 Container 21
3.1.2 Notebook 23
3.1.3 Slider 25
3.1.4 Value Set 26
3.1.5 Progress Indicator 27
3.2 Standard Dialogs 28
3.2.1 File Dialogs 28
3.2.2 Font Dialog 29
3.3 Information Presentation Facility 30

3.4 Summary 31

Chapter 4. Workplace Shell Components 33
4.1 CUA and the Workplace Shell 34
4.1.1 Icons 35
4.2 An Introduction to the WPS 35
4.2.1 The Desktop 36
4.2.2 Objects On The Standard Desktop 36
4.2.3 Objects And Views 37
4.2.4 Customizing The Workplace Shell Objects 38
4.2.5 Arranging Folders and Objects According to Tasks 40
4.3 Workplace Shell Objects 41
4.3.1 Device Objects 42
4.3.2 Container Objects 42
4.3.3 Data Objects 42
4.3.4 Reference Books 43
4.3.5 Program References and Shadows 43
4.3.6 Drives 43
4.3.7 The Shredder Object 44
4.3.8 Printer Objects 45
4.3.9 Templates 46
4.4 The LAN Independent Shell 47
4.4.1 Using the LAN Independent Shell 48
4.4.2 Main Functions 49
4.4.3 LAN Server and Novell Netware Support 50
4.5 Summary 51

Chapter 5. Using the Workplace Shell 53
5.1 WPS Navigation and Techniques 53
5.1.1 Mouse 53
5.1.2 Keyboard 53
5.1.3 Accelerators 54
5.1.4 Object Manipulation Techniques 54
5.2 Basic Operation of the Workplace Shell 56
5.2.1 Accessing a Context Menu 56
5.2.2 Opening a Window 56
5.2.3 Finding Open Windows 57
5.2.4 Creating A New Object 57
5.2.5 Creating a Shadow Object 58
5.2.6 Deleting and Undeleting an Object 58
5.2.7 Printing 58
5.2.8 Creating a Startup Environment 59
5.3 Advanced Operation of the Workplace Shell 60
5.3.1 Changing the Characteristics of an Object 60
5.3.2 Modifying the Context Menu of an Object 60
5.3.3 Changing the Default View on "Open" 61
5.3.4 Changing the Icon for an Object 62
5.3.5 Associating an Object with a Program 62
5.4 Giving OS/2 V2.0 the Look and Feel of OS/2 Version 1.3 64
5.5 Summary 65

Chapter 6. Installing and Supporting the Workplace Shell 67
6.1 Allocating Disk Space 67
6.1.1 Partitioning the Disk for OS/2 with the Workplace Shell 67
6.1.2 HPFS or FAT Format? 68

6.1.3 Keeping the Desktop Separate from the System 69
6.1.4 Moving the Print Spooler 69
6.2 Setting Up Programs and Files 70
6.2.1 Extended Attributes 70
6.2.2 EAs for Files Used by DOS Programs 70
6.2.3 Using Files Outside the WPS Directory Structure 71
6.2.4 Setting Up Programs in the Workplace Shell 71
6.3 Problem Determination and Resolution 73
6.3.1 Error Symptoms of a Malfunctioning Desktop 73
6.3.2 Shadow Copies of Programs 73
6.4 Backup and Restore with the Workplace Shell 74
6.4.1 Critical System Files 74
6.4.2 How to Back Up OS2.INI 74
6.4.3 Restoring a Backup Version of OS2.INI 75
6.4.4 Backup Programs 76
6.5 Using the Workplace Shell in a LAN Environment 76
6.5.1 Organization of a LAN Workplace 76
6.6 Workplace Shell Performance 79
6.7 Training Users to Use the Workplace Shell 79
6.7.1 Training and Desktop Configuration 79
6.8 Utilities for the Workplace Shell 80
6.8.1 Prevent Programs Restarting at IPL 80
6.8.2 File Transfer to a Host Session 81
6.8.3 Limiting a User's Access to Settings 82
6.8.4 Creating and Populating Folders 82
6.8.5 Adding New File Types 84
6.8.6 Removing WPS Objects 84
6.9 Customizing OS/2 V2.0 for the Inexperienced User 84
6.9.1 User Requirements 85
6.9.2 Operating System Set Up 86
6.9.3 Setting up the Users Work Area 87
6.10 Summary 89

Chapter 7. Presentation Manager and Workplace Shell Application Development 91
7.1 The Presentation Manager Application Model 91
7.1.1 Windows 91
7.1.2 Messages 94
7.1.3 Presentation Spaces and Device Contexts 97
7.1.4 Presentation Manager API Enhancements in OS/2 V2.0 97
7.2 The Workplace Shell Application Model 99
7.2.1 Workplace Shell Objects and Applications 99
7.2.2 System Object Model 100
7.2.3 Using Workplace Shell Classes 101
7.2.4 The Structure of a Workplace Shell Application 103
7.3 Writing PM Applications to Work with the Workplace Shell 103
7.3.1 The Workplace Shell and PM 104
7.3.2 How Much Can You Do with PM? 104
7.3.3 Migrating Existing Applications 104
7.4 Summary 109

Chapter 8. Workplace Shell Implementation 111
8.1 The Workplace Shell as an OS/2 Program 111
8.1.1 Workplace Shell and the System Object Model 112
8.1.2 Workplace Shell Object Types 112

8.1.3 Relationship of the Shell to the File System 113
8.2 Workplace Shell Objects 115
8.2.1 Folders 115
8.2.2 File System Objects 117
8.2.3 Shadows 119
8.2.4 Abstract Objects 119
8.2.5 Transient Objects 120
8.3 Workplace Shell Facilities 120
8.3.1 Object Registration 120
8.3.2 File Associations 121
8.3.3 Direct Manipulation 121
8.3.4 WPS Events/Messages 122
8.3.5 Persistence 123
8.4 Extended Attributes 124
8.4.1 Directory Extended Attributes 124
8.4.2 File Extended Attributes 125
8.5 The OS2.INI File 126
8.6 Multiple Instances of Objects 129
8.7 Summary 130

Appendix A. Using REXX in OS/2 V2.0 131

Appendix B. CUA Conformance in the Workplace Shell 133
B.1 Fundamental Items 134
B.2 Recommended Items 135
B.3 Other Items 136
B.3.1 Navigation 136
B.3.2 Emphasis 136
B.3.3 Mnemonics 136
B.3.4 Push Buttons 137
B.3.5 Miscellaneous 137

Glossary 139

Index 143

Figures

1. Presentation Manager Window 10
2. Presentation Manager Dialog Box 14
3. Presentation Manager Message Box 14
4. Clipboard Copy from an OS/2 Window into an OS/2 PM Editor 18
5. Container Control Used in Folder Window 22
6. Presentation Manager Notebook for Desktop Setting 24
7. Presentation Manager Notebook used in Master Help Index 25
8. Slider Control 26
9. Value Set Control 27
10. Progress Indicator Control 28
11. Standard File Dialog for "Open" 29
12. Standard Font Dialog 30
13. Workplace Shell Desktop Appearance 33
14. Different Views of the Same Objects 38
15. System Setup 39
16. Keyboard Settings Notebook 40
17. Workplace with Objects 41
18. Local Drives and LAN Drives 43
19. Shredder Object With A Folder For Deletion 44
20. Job List View of Print Object 45
21. Templates After Full Installation 46
22. Example of a Folder With User Templates 47
23. Example of a Menu Showing the Association 47
24. Tree View of the LAN Server 48
25. Different Objects - Different Functions 56
26. Expanded Desktop Menu 60
27. Setup of an Expanded Menu 61
28. Starting XCOPY From the First Line in CONFIG.SYS to Back Up the INI Files 75
29. Building Back Up History of the INI Files from STARTUP.CMD 75
30. A REXX Procedure To Prevent Programs Restarting 80
31. REXX Procedure for Host Upload 81
32. REXX Procedure to Create a New Folder 83
33. REXX Procedure to Add a Program to a Folder 83
34. REXX Procedure to Register a New WPS Class 83
35. REXX Procedure to Deregister a WPS Class 84
36. Workplace Shell Quotations Work Area 88
37. Presentation Manager Application Structure 92
38. Message Queues 95
39. Workplace Shell Class Hierarchy 102
40. An ASSOCTABLE Resource Script File Statement 108
41. Workplace Shell Class Hierarchy 112
42. Disk Structure Supporting the Workplace Shell 113
43. Drag/Drop - Physical Implementation 114
44. Relationship of Folder to Directory and OS2.INI File 116
45. Effect of Changing Description on HPFS file names 118
46. Effect of Copying Files on Filenames 118
47. Contents of Directory Extended Attribute (ICONPOS) 124
48. Contents of Directory Extended Attribute (CLASSINFO) 125
49. Contents of File Extended Attributes 125
50. Running Programs Stored in OS2.INI 127

51. Abstract Object Reference for Shredder in OS2.INI 127
52. Abstract Objects Contained in Folder 5506 128
53. Association Filters for File Extensions 128
54. Association Filters for File Types 129

Tables

1. Mouse Button Settings 53
2. Workplace Shell Object Persistence Summary 130
3. Fundamental Items 134
4. Recommended Items 135

Special Notices

This publication is intended to help customers and system engineers to understand and utilize the new features in Version 2.0 of OS/2. The information in this publication is not intended as the specification of the programming interfaces that are provided by OS/2 V2.0, Presentation Manager and the Workplace Shell. See the PUBLICATIONS SECTION of the IBM Programming Announcement for OS/2 Version 2.0 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Commercial Relations, IBM Corporation, Purchase, NY 10577.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

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

AIX
C/2
Common User Access
CUA
IBM
Micro Channel
OfficeVision
Operating System/2
OS/2
Personal System/2
PS/2
SAA
Systems Application Architecture
WIN-OS/2
Workplace Shell

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

286, 386, 486, SX are trademarks of Intel Corporation.
dBase IV is a trademark of Ashton-Tate, Inc.
Intel is a trademark of Intel Corporation.
Lotus, 1-2-3 are trademarks of the Lotus Development Corporation.
Microsoft, Windows are trademarks of Microsoft Corporation.
Novell, Advanced Netware are trademarks of Novell, Inc.
SY-TOS is a trademark of Sytron Corporation.
Smalltalk/V is a trademark of Digitalk, Inc.
WordPerfect is a trademark of WordPerfect Corporation.

Preface

This document gives a description of the functions and facilities provided by the Presentation Manager and Workplace Shell in OS/2 Version 2.0 and their implementation.

The document discusses the migration of Presentation Manager applications from previous versions of OS/2 and the considerations for deciding whether to implement new applications as Presentation Manager or as Workplace Shell applications.

This document is intended for planners and technical support personnel who require an understanding of the function provided by the Presentation Manager and Workplace Shell in OS/2 Version 2.0, and the implementation of that function.

Sample code relating to the Workplace Shell is available in electronic form via CompuServe** or through a local IBM Support BBS, as package RB3774.ZIP. IBM employees may obtain the code examples from the GG243774 PACKAGE on OS2TOOLS.

This sample code was developed primarily for *OS/2 Version 2.0 - Volume 4: Application Development* and therefore has the same package name.

The document is organized as follows:

- *Chapter 1, "Introduction to the Presentation Manager and Workplace Shell"* provides a brief introduction to the topics covered in this document.

  This chapter is recommended for all readers of the document.

- *Chapter 2, "Presentation Manager Components"* provides an overview of the Presentation Manager graphical user interface, describing the components of the interface and the interaction between them.

  This chapter is recommended for those readers who are not familiar with Presentation Manager and its operation.

- *Chapter 3, "New Presentation Manager Features"* describes the new controls, dialogs and functions provided by Presentation Manager.

  This chapter is recommended for those readers who are familiar with Presentation Manager and wish to learn how it has been enhanced in OS/2 Version 2.0.

- *Chapter 4, "Workplace Shell Components"* gives a brief introduction to the Workplace Shell and its role in CUA. It provides an overview of the Workplace Shell user interface components, including both local resources and those available through the LAN Independent Shell.

  This chapter is recommended for those readers who are not familiar with CUA concepts or the Workplace Shell facilities.

- *Chapter 5, "Using the Workplace Shell"* explains how the components of the Workplace Shell work and gives guidance on how to use them effectively in a number of user scenarios.

  This chapter is recommended for those readers who are not familiar with the Workplace Shell and its use.

- *Chapter 6, "Installing and Supporting the Workplace Shell"* gives guidance on the installation and configuration of the Workplace Shell and some advice on sorting out problems that may arise.

  This chapter should be read by anyone planning to install or to support OS/2 Version 2.0 Workplace Shell.

- *Chapter 7, "Presentation Manager and Workplace Shell Application Development"* provides a conceptual introduction to the Presentation Manager and Workplace Shell application models, describing the major components of the two styles of application and their interaction. It also discusses the question of how far a Presentation Manager application can integrate with, and make use of, Workplace Shell facilities such as printers and the shredder.

  This chapter is recommended for readers considering any application development for OS/2 Version 2.0

- *Chapter 8, "Workplace Shell Implementation"* discusses how Workplace Shell is implemented on OS/2 and Presentation Manager. In particular, this chapter describes Workplace Shell's use of such OS/2 facilities as Extended Attributes and OS2.INI for its data.

  This chapter is recommended for those readers who will be supporting OS/2 Version 2.0 and for whom an understanding of the implementation of Workplace Shell should prove invaluable when problems arise.

- *Appendix A, "Using REXX in OS/2 V2.0"* provides information on using Workplace Shell commands in REXX and gives an overview of a few of the most important commands.

- *Appendix B, "CUA Conformance in the Workplace Shell"* discusses the degree to which the Workplace Shell is compliant with CUA 91. It also provides some guidance to application developers on which behaviors are determined by the WPS and which should be set in the program.

Related Publications

The following publications are considered particularly suitable for a more detailed discussion of the topics covered in this document.

Prerequisite Publications

- *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*, SC34-4289

Additional Publications

- *OS/2 Version 2.0 - Volume 1: Control Program*, GG24-3730
- *OS/2 Version 2.0 - Volume 2: DOS and Windows Environment*, GG24-3731
- *OS/2 Version 2.0 - Volume 4: Application Development*, GG24-3774
- *OS/2 Version 2.0 - Volume 5: Print Subsystem*, GG24-3775
- *OS/2 Version 2.0 Remote Installation and Maintenance*, GG24-3780
- *IBM Systems Application Architecture CUA Advanced Interface Design Reference*, SC34-4290
- *IBM Personal Systems Developer, Winter 1992*
- *IBM Icon Reference Book*, SC34-4348
- *IBM OS/2 Version 2.0 Application Design Guide*, 10G6260
- *OS/2 2.0 Programming Guide Volume II*, 10G6494

Chapter 1. Introduction to the Presentation Manager and Workplace Shell

OS/2* Version 2.0 brings new levels of computing power to users' desktops, and is designed to deliver this power to a very wide range of users, many of whom may have little experience with computers.

Power and sophistication could mean complexity for the user, but OS/2 Version 2.0 has been provided with an advanced user interface designed to provide users with this power in a way that is easy to learn and, as far as possible, intuitive, enabling them to make the most of their investment in hardware and software.

The components of OS/2 Version 2.0 that provide this user interface are Presentation Manager* and the Workplace Shell*. These are the subjects of this document.

This introductory chapter:

- Describes a vision of what is made possible with the Workplace Shell
- Describes what Presentation Manager and the Workplace Shell are and how they fit with one another and the rest of OS/2
- Includes a summary of the enhancements to Presentation Manager.

1.1 The Vision

Those who have worked with computers for many years are very happy to talk about files, programs, directories, records, spoolers, etc. The inexperienced computer user is not. We are used to the idea that when editing a document we must first load the data from disk into memory and later, when we have finished editing it, save it back onto disk. The new computer user is not.

While we are using a computer we can visualize what is going on in the computer - programs being loaded into memory, print files being written to disk before being printed, and the whole hierarchy of disks, directories and files. The user interface, for us, is a means by which we control these processes. There have been good user interfaces and bad user interfaces, but so far they have all been designed to allow the knowledgeable user to control a system he understands to some degree. Becoming proficient in the use of a personal computer meant understanding what it does internally.

With Presentation Manager and the Workplace Shell a new user can enjoy much of the function of OS/2 and its applications, without needing to understand these things. He does not even need to think of what he sees on the screen as an interface - that would imply that he is aware that there is something beyond it to which he is interfacing. What he sees on the screen is all he needs to know about. And what he sees there are representations of everyday items such as folders, documents, an alarm clock, a shredder, and printers.

These items - known as *objects* and represented on the screen by little images known as *icons* - behave in familiar ways; for example to destroy a document a user would probably guess that he had to put it through the shredder. With Workplace Shell this is almost exactly what he would do - using the mouse he

would drag the icon representing the document onto the icon representing the shredder. Having discovered that, he might then guess that moving a document to a printer would cause it to be printed, or moving it onto a mailbox icon would result in its being sent to another user.

Of course, if all you could do with OS/2 Version 2.0 were the same things you could do without it, there would be no need to have a computer at all. The point is, of course, that though the objects with which the OS/2 user works do familiar things in familiar ways, they also have capabilities far beyond their real-life counterparts. Consider, for example:

- A folder that will, on request, automatically give you all the monthly cashflow summaries, from its contents of 1000 miscellaneous papers
- A pad of order forms that never runs out
- A desk on which you can leave all your papers at the end of the day and be sure that they are still as you left them when you come in the next day, with no fear of their being moved by the cleaners, or of confidential documents being removed
- A sheet of paper which automatically corrects the spelling of everything you write on it.

  (This would be the Workplace Shell equivalent of a word-processor with a spellcheck function - the difference is that the user need not know of the existence of any word-processing package; to him, the spell-checking capability is a characteristic of the document itself. This is the essential difference between object-oriented and application-oriented interfaces.)

OS/2's Workplace Shell can do all this and, moreover, do it in a way that the inexperienced user will find a straightforward and natural extension to the real-life behavior of the objects concerned. Furthermore, the Workplace Shell allows application developers to extend the range of objects available to the user to include many more sophisticated or specialized types than those provided as standard with OS/2.

1.1.1 The Workplace Shell User with Older Applications

In reality, relatively few users will be able to work in this way all the time. The chances are they will want to use some applications that were written for OS/2 1.3, or for DOS or Windows. These generally work in ways that are not consistent with the purely object-oriented interface described above.

For these users the Workplace Shell provides a very flexible desktop environment from which to launch their non-object-oriented applications. For example, the *Program Reference* allows the user to represent any program as a Workplace Shell object. This lets it be placed on the desktop or in any convenient folder and started when required. Furthermore, the Workplace Shell provides the ability to associate any program with particular types of data files, so that the user need only open the data file for the associated program to be automatically invoked. This provides an object-oriented technique for starting some programs that were not written with this in mind.

1.1.2 What About the Experienced PC User?

Although the Workplace Shell can provide the inexperienced user with this ideal working environment requiring no knowledge of how OS/2 works, there are many users who are already quite comfortable with basic computing concepts. These users could feel frustrated if they were denied the ability to access the files, directories and programs with which they are familiar.

The Workplace Shell does not prevent this. For example, it provides the *Drives* object, offering similar function to the file managers of OS/2 Version 1.3 or Microsoft Windows**. The *Program Reference* object class provides a way to install and run ordinary - non-Workplace Shell - programs. OS/2 and DOS command prompts may be invoked, allowing the use of commands some of which date back over 10 years to the first releases of DOS. Workplace Shell is also so flexible that a user can set it up to look and behave very much like OS/2 Version 1.3 or Windows, if that is what is preferred.

So, experienced PC users may find the Workplace Shell unfamiliar at first, and may not even fully understand the point of it, but they have the option of working in whatever way they choose - using a command prompt if they want to, or perhaps an interface similar to OS/2 Version 1.3 or Windows. In time, they may learn to to think in a less computer-oriented way and start using the additional productivity features of the Workplace Shell.

1.2 What is Presentation Manager?

Presentation Manager provides a windowed graphical user interface for OS/2 applications. The user interacts with these applications using interface constructs controlled by Presentation Manager, with either the keyboard or the mouse. For example, when a program asks you to enter a short piece of text such as a file name, you type it into a Presentation Manager control known as an *Entry Field*, when a program wishes to display a scrollable list it uses a control known as a *List Box*, and so on. Presentation Manager has two basic objectives:

- To provide a *consistent* user interface for both the operating system and applications, so that users may more easily interact with a number of applications, without the need to learn different sets of interface rules.
- To provide an *intuitive* interface for the end user, in order to ease the task of learning applications and reduce the amount of formal training required by encouraging learning through exploration.

The Presentation Manager user interface is based on **icons**, which are used to represent the objects with which the user wishes to work, and **windows**, which typically provide views of the contents of those objects. For example, an icon might represent a document which the user wants to modify; in order to do this he would open a view of the document in the form of a window displaying the text in an editable form. When he has finished working on the document he would close the view.

The basics of the Presentation Manager user interface are described in Chapter 2, "Presentation Manager Components."

The implementation of Presentation Manager under OS/2 Version 2.0 has been enhanced over previous versions of OS/2. Presentation Manager itself has been rewritten to take advantage of the 32-bit functions available in Version 2.0,

resulting in improved performance, particularly for complex graphical applications, and several new user interface controls and standard dialogs have been introduced in conformance with the 1991 Systems Application Architecture* (SAA*) Common User Access* (CUA*) Workplace Environment. The Information Presentation Facility (IPF) has also been enhanced.

1.3 What is the Workplace Shell?

The Workplace Shell is both a user interface to OS/2 itself and an application environment in which user-written programs can integrate themselves with one another, and with OS/2's user interface. This section describes these two aspects of the Workplace Shell.

1.3.1 Workplace Shell as an Operating System Shell

Any operating system must provide a user interface that allows the user to make use of the functions of the system, for example to start programs, manage files and so on. This interface is known as its **shell**, and in some operating systems is no more than a command line and a set of commands; the user has to know the correct commands to start programs, manipulate files, tailor the system, and so on. An example of such a shell is the command line interface of DOS 3.3.

OS/2 1.1 had a shell that used the Presentation Manager interface. This consisted of a collection of utility programs allowing for starting, stopping and switching between programs (the Desktop Manager), manipulating directories and files (the File Manager), altering system settings (the Control Panel), and managing printers, print queues and the spooler (the Print Manager).

This shell provided a degree of consistency for the user through its use of Presentation Manager. However, it was still no more than a collection of separate programs, each working in its own way and having to be learned by the user. Furthermore, it made no attempt to hide from the user the complexities of the operating system, requiring him to understand concepts such as programs, files, and directories. Nevertheless, this shell remained, with only minor changes, through OS/2 Releases 1.2 and 1.3.

OS/2 Version 2.0 introduces a new shell - the Workplace Shell - which takes this development one stage further. It provides a shell that is more consistent, and allows the inexperienced user to remain largely ignorant of the operating system itself. The Workplace Shell implements a type of interface known as an **Object-Oriented User Interface** because with it the user interacts with icons representing familiar objects, such as documents, printers, shredders, and folders. The user's attention is focused on these objects, rather than on the programs and files that lie behind them, as with most other kinds of user interface.

The Workplace Shell provides all the function of the old OS/2 Version 1.3 shell in a more consistent and easily learned way, while at the same time providing much greater flexibility for the user to organize his work in the way that suits him.

1.3.2 Workplace Shell as an Application Environment

The Workplace Shell provides a programming environment, in which suitably written programs can add user-defined object types to those provided with the system. Such objects may simply be modified versions of the supplied object types - such as a new type of folder that requests a password before the user is allowed to open it - or object types related to specific productivity applications, such as a *spreadsheet* object, or even object types that are specific to a particular user's business, such as *Customer*, *Order* or *Motor Insurance Policy*.

It has been found that a natural and productive way to implement this kind of user interface is to use object-oriented programming techniques. The Workplace Shell itself is written in this way, using a new component of OS/2 V2.0 called the **System Object Model** (SOM). SOM is a set of tools and APIs that allows object-oriented programs to be written in a mixture of programming languages, including languages that are not themselves object-oriented, such as C. All the Workplace Shell object types - folders, data files, printers, etc. - are implemented as SOM objects.

1.3.3 WPS Objects versus SOM Objects

Please note that in this book, and in most other discussion of the Workplace Shell, we use the word **object** in two quite distinct, but closely related, senses:

- A Workplace Shell object is something that a user interacts with and manipulates, and is represented by an icon. It normally represents some real-world item that the user understands, such as a printer or an order form.
- A SOM object is a programming construct and is the basis of SOM's implementation of object-oriented programming (OOP). It may represent something quite meaningless to the user such as a database table, or an APPC conversation; fortunately, a user would not normally know anything about SOM objects.

Workplace Shell objects are implemented as SOM objects, but SOM objects do not necessarily have anything to do with Workplace Shell; indeed a SOM object may not have any visible form at all, being no more than a piece of program code and data.

It should normally be clear from the context which sense of the word *object* is intended.

1.4 Presentation Manager Enhancements in OS/2 Version 2.0

This section introduces the Presentation Manager enhancements in OS/2 Version 2.0, all of which are discussed in more detail later in this document.

1.4.1 New Controls and Dialogs

A number of new control window classes are provided under OS/2 Version 2.0. These are primarily used to aid in the implementation of the Workplace Shell and the 1991 SAA CUA Workplace Environment, and are available for use by application developers. They are:

1.4.1.1 Container

The purpose of the container control is to hold other objects within the Workplace Shell, in order to allow grouping of objects on the desktop in a logical manner determined by the end user. A container control may display objects in various formats or views, where each view displays different information about the objects.

1.4.1.2 Notebook

The notebook control provides a method for organization and navigation of user dialogs, where the required information and prompts may not be easily displayed in a single dialog box. A notebook control looks like a multi-page notebook, with a dialog on each page. The user can interact with the top page, and can move to the others either page-by-page or directly, by using tabs attached to the edges of the pages.

1.4.1.3 Slider

The slider control looks rather like the slider controls found on some audio equipment, and is used where a value must be chosen from a continuous but finite range of settings. It is ideal for setting approximate values and properties, particularly analog values which are not easily enumerated or expressed in other ways. The slider indicates a particular quantity and the range of allowable values for that quantity.

1.4.1.4 Value Set

The value set control is similar in function to the radio button control implemented in previous versions of OS/2, but provides additional flexibility in that it may be used to display a set of values in graphical, numeric, or textual format, whereas the radio button is limited to textual format only. As with a radio button, selecting one item in the set deselects any previously selected value.

1.4.1.5 Progress Indicator

The progress indicator control is used to display the progress of a long-running operation, such as file transfer or disk backup/restore, by means of a hollow bar that fills with color from one end like a thermometer.

1.4.2 Information Presentation Facility Enhancements

A number of functional enhancements have been made to the Information Presentation Facility (IPF), which provides context-sensitive help panels and online documentation. These enhancements are as follows:

- The ability to predefine default sizes for help panels; these sizes are used unless overridden by the calling routine.
- The ability to define hypertext and hypergraphic links across multiple concatenated help or documentation files.
- Support for multiple viewports within a panel, allowing different types of complimentary information to be displayed together but to be manipulated independently.
- Dynamic data formatting, allowing help text or explanatory information to be formatted and placed into a help panel at run time, thereby allowing help information to be closely tied to the current user action.
- Support for tables in help files and online documentation.
- Support for multiple fonts and text sizes.

These enhancements are described in greater detail in Chapter 3, "New Presentation Manager Features."

1.5 Programming Environment

The Presentation Manager programming environment under OS/2 Version 2.0 has changed very little from that provided under OS/2 Version 1.3. Presentation Manager applications written for previous versions of OS/2 will execute without modification under Version 2.0.

A high degree of source code compatibility is also provided which allows such applications to be recompiled for execution in a 32-bit environment, providing enhanced performance, with only very minor modifications in most cases.

A number of new API functions have been added to Presentation Manager under OS/2 Version 2.0, providing additional function to applications. These include the **WinPopUpMenu()** function which allows the creation of context menus and a number of other functions to simplify tasks such as checking or unchecking menu items, that were previously achieved by means of PM messages, some enhancement to the *Gpi* graphical functions, and some other minor changes to function names etc.

A number of standard CUA-conforming dialogs for file and font manipulation have been included within the operating system, removing the need for application developers to write these commonly needed functions as part of their application code.

Presentation Manager applications under OS/2 Version 2.0 may also mix 16-bit and 32-bit executable modules, in the same way as other applications. The operating system provides a "thunk" layer for the PM APIs as it does for the kernel APIs. Some additional considerations arise, however, when registering windows between environments and when passing pointers as message parameters, due to the difference in addressing schemes; this is handled automatically for the standard message classes. Functions are provided to enable programmers to implement thunks for their user-defined messages classes. For a general explanation of thunks see *OS/2 Version 2.0 - Volume 1: Control Program* and for examples of their use see *OS/2 Version 2.0 - Volume 4: Application Development*.

In order to implement application objects that integrate fully with those provided with the Workplace Shell one must write parts of one's application in a new way, using the System Object Model. The System Object Model provides a language for defining object classes, their methods and instance data, known as **Object Interface Definition Language** (OIDL) and some utility programs to assist in the generation of the language source files that are needed to build those object classes. Although currently only implemented for the C language, the System Object Model has been designed to enable object classes to be written in a variety of both object-oriented and non-object-oriented languages.

All the object classes that the user sees when using Workplace Shell, such as folder, data file and printer, are implemented internally as SOM classes (wpFolder, wpDataFile, wpPrinter, etc.). By sub-classing these and other Workplace Shell classes, programmers can develop new classes for the particular needs of his users, inheriting all the useful instance data and methods while adding to or replacing them with those that their new objects require.

Programming for the Presentation Manager and Workplace Shell environments under OS/2 Version 2.0, including the use of the System Object Model and the Workplace Shell classes, is described in detail in *OS/2 Version 2.0 - Volume 4: Application Development*.

1.6 Summary

Presentation Manager provides the strategic graphical-based user interface and programming environment for OS/2 Version 2.0. The implementation of Presentation Manager under Version 2.0 has been significantly enhanced over previous versions of OS/2 by exploiting the 32-bit application environment, and by providing additional controls and functions. These can make development more productive for the programmer and the interface more consistent for the user.

The object-oriented Workplace Shell, which replaces the PM shell of previous releases, implements the SAA CUA Workplace Model, both for itself and for applications written to exploit the programming environment it provides. Such applications behave in a more intuitive manner, allowing the user to spend less time learning system-specific operations and more time concentrating on the work tasks to be performed. The functions previously implemented in desktop utilities have been combined within the Workplace Shell, insulating the user from the complexities of the system.

A number of new Presentation Manager control windows and functions have been added under Version 2.0, providing additional function to applications and enhancing the flexibility of user dialogs by providing additional mechanisms for displaying information and receiving user input. Standard dialogs have also been provided for common file and font manipulation functions, ensuring SAA CUA conformance and removing the need for applications to include such functions within the application code.

The Presentation Manager programming environment has changed very little under OS/2 Version 2.0. Applications written and compiled for previous versions will normally run with no changes. For applications to take full advantage of the 32-bit environment and new Presentation Manager functions, some changes to the source code are required, though these are mostly minor.

A new, object-oriented, programming model is introduced in the System Object Model, which, in conjunction with the object classes supplied by the Workplace Shell, enables application objects to be developed that integrate seamlessly with the Workplace Shell. Applications written in this way can inherit useful behavior from the supplied Workplace Shell object classes and add function unique to the requirements of a user's business.

In general, Presentation Manager under OS/2 Version 2.0 provides improved performance through the use of a 32-bit graphics engine and the Workplace Shell provides improved usability. The redesigned shell with its more intuitive interface helps to insulate the user from the inherent complexity of OS/2, allowing the user to concentrate on the task being performed. The result is improved productivity with reduced training requirements.

Chapter 2. Presentation Manager Components

The OS/2 Presentation Manager user interface was introduced in OS/2 Version 1.1, replacing the textual interface provided by OS/2 Version 1.0. Presentation Manager provides a graphical, windowed presentation environment that lets a user execute and view multiple applications on the screen at the same time. The user can interact with those applications using an event-driven, object-action user interface which conforms to the guidelines laid down in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*.

The Systems Application Architecture CUA component has two major objectives:

1. To provide a *consistent* interface to applications, in order that users may more easily interact with a variety of applications and operating system environments without the need to learn and remember a different set of interface rules and guidelines for each application or system.
2. To implement the aforementioned consistency objective in such a way that the interface to applications is highly *intuitive*, in order to minimize the amount of formal application training required by affording an easy to use interface which encourages users to learn by exploration.

Both of these objectives can be fulfilled by the Presentation Manager components. This chapter will outline some of the major features of the Presentation Manager user interface, for those readers who may be unfamiliar with it.

The Workplace Shell is not a complete implementation of the CUA 91 **Workplace Environment**. Differences are discussed in Appendix B, "CUA Conformance in the Workplace Shell" on page 133. In this chapter, we use standard Presentation Manager terminology. This is consistent with the terminology used in the accompanying *OS/2 Version 2.0 - Volume 4: Application Development*. It is also more correct than using CUA terminology with objects which are not CUA-compliant.

2.1 Windows

Presentation Manager applications revolve around the concept of **windows**. A window appears as a rectangular area on the screen, in which information may be displayed and user input entered.

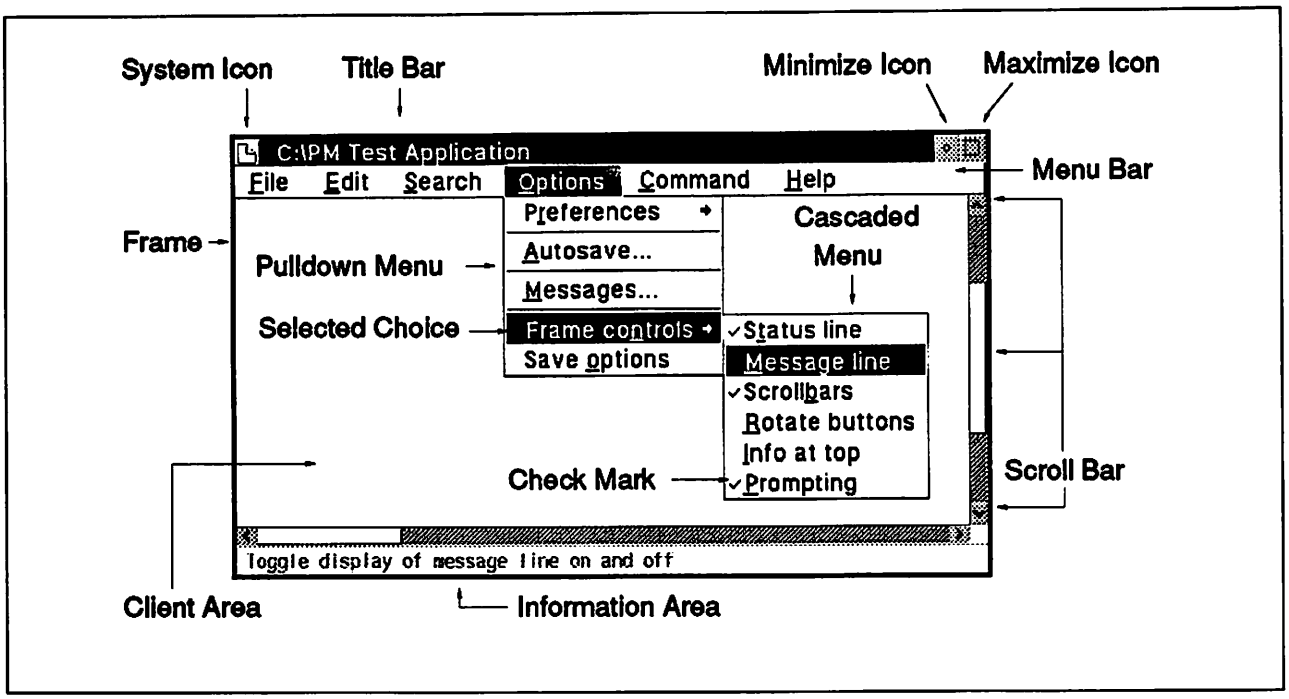

Figure 1. Presentation Manager Window

An application may create multiple windows, and multiple windows created by one or more applications may be displayed concurrently on the screen.

While the Presentation Manager interface is graphics-oriented, the information input and output through a window need not be graphical; text may be manipulated in windows without the complication of using graphics fonts. While a mouse is recommended for user interaction in the Presentation Manager environment, many Presentation Manager functions can be achieved by the use of the keyboard only.

Windows are displayed on the screen overlaying a background known as the **desktop**. This desktop may be altered to any color, or a bitmapped image may be displayed, using utilities provided with Presentation Manager as part of OS/2 Version 2.0.

An application may create multiple windows; the desktop may simultaneously display multiple windows created by multiple applications, and these windows may be updated concurrently by their parent applications, due to the multitasking nature of the OS/2 operating system. However, since OS/2 is a *single-user* system, the user provides input to only one window at a time. This window is said to possess the **input focus**. The user may switch the input focus from one window to another by pointing to the desired window with the mouse and pressing mouse button 1, or with the keyboard using the Ctrl+Esc or Alt+Esc key combinations.

The use of the mouse button depends on whether the user is left- or right-handed. It can be selected during the installation process of OS/2 Version 2.0. In this document we conform to the CUA convention of using mouse button 1 (MB1) for "select" and mouse button 2 (MB2) for "drag" functions. More information on this is provided in 5.1, "WPS Navigation and Techniques" on page 53.

When many windows are displayed on the desktop at one time, the desktop can become cluttered and almost unworkable. To avoid this situation and remove

unwanted windows from the screen without terminating their associated applications, Presentation Manager allows the user to **minimize** a window. When a window is minimized, it is removed from the desktop and its **icon** is placed in the *Minimized Window Viewer* folder. See 2.5, "Icons" on page 16 for a further discussion of this.

Clicking MB2 on an icon causes Presentation Manager to display a context (or pop-up) menu. This lets the user restore, move or close the window (that is, terminate the application). If the user "double clicks" MB1 on the icon, Presentation Manager will restore the window to its previous size and position on the screen, without displaying the menu.

In order to display a window to the full size of the display screen, a user may **maximize** a window. The window is then resized to the maximum defined by the program. For some programs this will be the whole screen. A window may also be restored to its former size and position on the screen. See 2.1.1, "Frame Area" for a description of the way in which this is achieved.

A window on the screen consists of two distinct areas; the **frame** area and the **client** area. The frame area allows the user to manipulate the window (that is, resize the window, move the window on the screen, etc.) while the client area is used by the application to display information and solicit user input.

2.1.1 Frame Area

The frame area contains a number of other windows, such as the system menu, title bar and menu bar, which allow the user to perform window manipulation functions. These control windows partly conform to the definitions laid down in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*. They are illustrated in Figure 1 on page 10 and explained below:

Sizing Border A standard window has a sizing border, which allows the window to be **sized** (that is, made smaller or larger) using the mouse or a function-key sequence. When the mouse pointer is moved over the sizing border, the pointer changes from the standard (arrow) pointer to a special sizing pointer. The user clicks and holds one of the mouse button while moving the mouse, and the window is sized accordingly.

Title Bar A standard window has a title bar, which performs two functions. Firstly, it identifies an application or window to the user. Secondly, it acts as a "handle" whereby the window may be repositioned on the screen. The user moves the mouse pointer over the title bar, clicks and holds mouse button 2 while moving the mouse, and the window moves accordingly.

The 1991 SAA CUA guidelines stipulate that the title bar for a window should contain both the system menu icon and a title bar icon for that window.

Menu Bar The main window of an application also has a menu bar, which acts as a primary menu for the application. Entries in the menu bar are selected by pointing with the mouse pointer and clicking a mouse button. Each menu bar entry is associated with a **pull-down menu**; (a list of actions associated with the entry), which appears when the bar entry is selected. The pull-down menu provides submenus for the menu bar. Multiple levels of pull-down menus may be used in an application, although the

number of levels is normally minimized for the sake of simplicity.

Minimize Icon In the top-right corner of a standard window, two icons are displayed; the left one is the minimize icon. When it is selected, the window itself is reduced to an icon. The icon may be generated by the application controlling the window, or may be a default icon supplied by the Presentation Manager.

Maximize Icon The maximize icon is displayed in the top-right corner of a standard window along with the minimize icon. When selected, the maximize icon causes the window to be resized to the maximum defined by the program. Other windows on the screen may be made invisible (although they are still present, logically "behind" the maximized window). If a maximized window is explicitly resized using the sizing border, other windows may become visible again. A maximized window may be restored to its former size and position on the screen by using the **restore icon** (see below).

Restore Icon In a maximized window *only*, the restore icon replaces the maximize icon. When selected, this icon causes the window to return to the size and position it occupied before it was maximized.

Small Icon In the top-left corner of a standard window, a small icon is displayed. This is a minimized version of the object icon. When selected using the mouse pointer, it displays a system menu with move, size, minimize, maximize and restore options. This menu provides an alternative to the use of the frame area border, title bar and icons for these operations.

Scroll Bars If the information in the client area is too big to fit into the size of the window on the screen, scroll bars should be created to allow the user to scroll through the information. A scroll bar can be either vertical or horizontal. It consists of a slider with arrow icons at each end of the bar. Clicking the mouse on an arrow scrolls one line in the indicated direction, while clicking on the bar between the arrow and the slider scrolls one page in the indicated direction. Clicking and holding down MB1 while dragging the slider up or down the scroll bar results in continuous scrolling.

Note that these facilities are common to most, if not all Presentation Manager applications, and their processing is handled by Presentation Manager rather than by the application. Thus the user is provided with a consistent interface for interacting with and manipulating windowed applications on the screen, which mostly conforms to the guidelines laid down in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*.

2.1.2 Client Area

The client area of a window contains information which is specific to the application, and thus the contents and layout of the client area will vary from one application to the next. To preserve the ideal of a consistent user interface, however, Presentation Manager provides a number of standard **control windows** which may be used to display and receive information to and from the user.

For further information about control windows see 2.4, "Control Windows" on page 15.

2.1.3 Parent and Child Windows

An application will normally have one main window and may have one or more **child windows**, subordinate to this main window. The main window is known as the **parent** of these child windows, and the child windows themselves may be the parents of other child windows. An application may thus have a hierarchy of windows. Too deep a hierarchy, however, can be very confusing to the user. Windows which exist at the same level in the hierarchy and have the same parent window are known as **siblings**.

While multiple windows may be visible on the screen at any time, the keyboard can only be associated with one window at a time; this window is said to have "input focus." Input focus is normally attributed to a control window on the client area, such as an entry field or list box. The window that contains the control with input focus, is then called the "active window." Presentation Manager automatically indicates this by making the active window the top window and changing the colors of its sizing border and title bar. The active window and input focus can be easily changed by the user. The same technique is used for both windows and controls; move the mouse over the desired window and select it using mouse button 1. Alternatively, a user may key combinations, such as ALT+ESC or ALT+TAB, to move between windows.

2.2 Dialog Boxes

A **dialog box** is a special type of window for use in certain circumstances where an additional interaction with the user is required. Dialog boxes may be of two distinct types; **modal** or **modeless**.

A dialog box is created as a short-lived window to receive and/or display a particular set of information required for the correct performance of an action by the application.

With a modal dialog box, the user may not interact with another window until the dialog is complete. This modality may be within an application (preventing interaction with other windows controlled by this application program, while allowing interaction with other applications in the system) or system-wide (preventing interaction with any other window until the dialog is complete).

With a modeless dialog box, the user may continue to work with the primary application window, for example using the "Find" window with the *OS/2 System Editor*. Modeless dialogs are preferred by CUA.

One difference between a dialog box and a standard window is that the dialog box may not be sized on the screen, nor may it be maximized or minimized except in certain circumstances. This property of a dialog box is crucial in maintaining the integrity of the dialog with the end user. Note that although a dialog box is not sizable, it may be moved on the desktop.

An example of a dialog box is shown in Figure 2 on page 14.

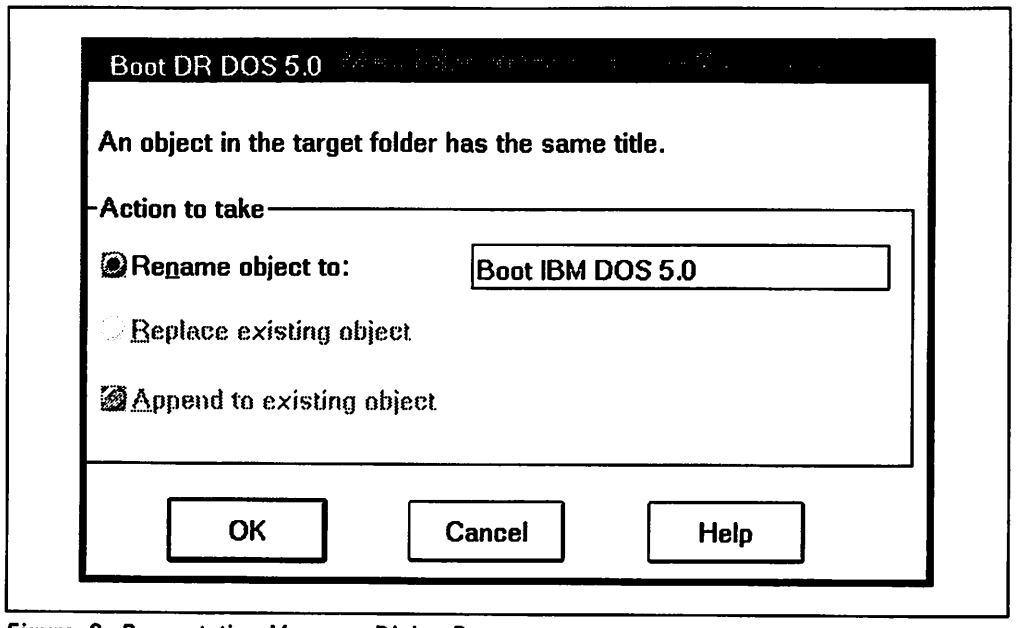

Figure 2. Presentation Manager Dialog Box

For example, if a user selects "Open" from the "File" pull-down menu, the application requires the name of the file before it can be opened; this information will be requested using a modal dialog box. For confirmation with the user, a **message box** may be used as an alternative to a dialog box; see 2.3, "Message Boxes" for further information.

2.3 Message Boxes

A message box is a short-lived window, similar to a dialog box, which is used to display a message to the user and to receive acknowledgement and a simple decision from the user. A message box is used to inform the user of an event in situations where the information to be conveyed is relatively simple, and the response returned by the user is limited to a single choice from a finite set of options. In such a case, the range of facilities provided by a dialog box is not required. Message boxes and their uses are discussed in detail in *OS/2 Version 2.0 - Volume 4: Application Development*. An example of a message box is given in Figure 3.

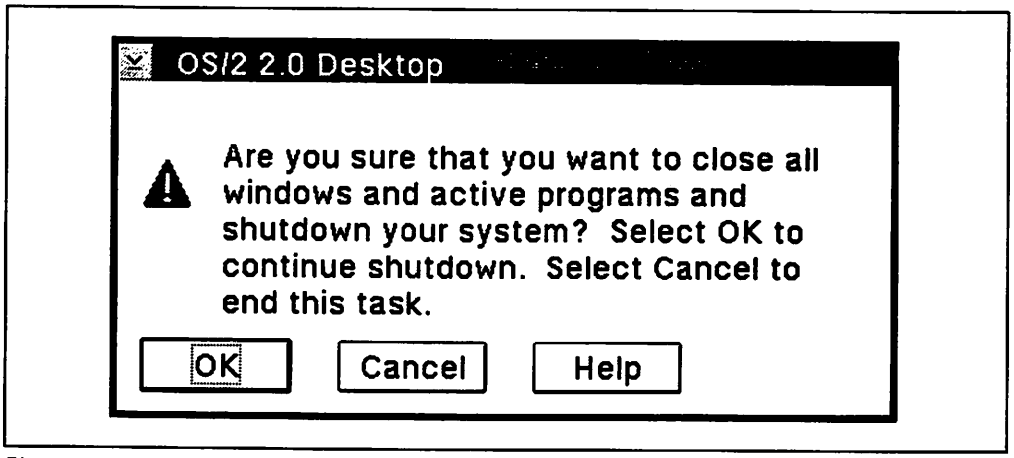

Figure 3. Presentation Manager Message Box

A message box displays an application-supplied text string, along with one or more push buttons such as "OK," "Cancel," "Help," etc. The user selects one of these buttons in response to the text displayed in the message box.

As defined in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*, a message box is a modal dialog with the user, since the user is required to acknowledge and act upon the message before continuing interaction. Presentation Manager allows both **application-modal** messages, which prohibit the user from interacting with any other window in the same application before responding to the message, and **system-modal** messages, which block interaction with any other window in the system until the message has been acknowledged.

2.4 Control Windows

The window areas and dialog boxes contain a number of **control windows** to display and receive information. These control windows are in fact specialized classes of window defined by Presentation Manager. The various types of control window are briefly explained below; complete definitions of the appearance and behavior of each type of control window are given in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*.

Static Text This is constant text, displayed in the dialog box for information purposes only; such text does not change from one invocation of the dialog to the next.

Entry Field This is an area, of a defined size, where a user may enter alphanumeric or numeric data. An application may also place a default entry in an entry field to simplify the user's task of entering information; the user may then edit the existing data or replace it with new data.

Entry fields may be single-line or multi-line.

List Box This is an area which contains a list of items, one or more of which may be selected by the user. If a list contains more information than will fit into the designated area, a list box may contain vertical and/or horizontal scroll bars.

Combo Box The combo box or **prompted entry field** is a combination of the entry field and list box types; by default, an entry field is displayed into which the user may enter text. At the right-hand side of the entry field however, is an icon which when selected, causes a *drop-down* list box to appear below the entry field, containing a list of valid entries. The user may enter a value directly into the entry field or, if unsure, may use the list box to select a valid entry.

Check Box This is a square area, surrounded by a border with accompanying text, which denotes an option that may be toggled on and off by the user. For instance, in a text search the option to ignore case may be implemented as a check box.

Radio Button Radio buttons are small round areas which are usually displayed in groups; a group of radio buttons denotes a series of mutually exclusive options from which the user may select one option only. Making a selection immediately deselects any previous

choice. The selected option in a group of radio buttons is indicated by a dark center in the button.

Push Button This is a square or rectangular area containing some brief text and surrounded by a border. It denotes an option which may be selected for immediate action. Selections such as "Enter" and "Cancel" to complete or cancel a dialog are normally implemented in this way.

Spin Button The spin button is used to display a currently selected option from a finite range of options. The items displayed in the spin button control are textual, and may represent any alphanumeric item. It consists of a single-line entry field, and up and down arrows that are stacked on top of one another.

Slider This is a scale with an index which may be moved along the scale using the mouse or keyboard. The slider allows the selection of a value from a contiguous set of values.

Value Set This is similar in concept to a group of radio buttons, except that items within the group may be icons, bitmaps or color patches as well as text strings. The value set control window allows the selection of a non-textual item from a set of mutually exclusive options.

Container A container acts as a repository and provides a mechanism to organize the desktop to suit the requirements of the user. Its basic function is to hold objects such as files, applications and devices represented by icons.

Notebook A notebook is a visual component that organizes information on individual pages so that a user can find and display that information quickly and easily. This component simulates a real notebook but improves on it by overcoming the real notebooks natural limitations.

These control windows are defined as child windows with the main window as their parent.

Note also that control windows are not restricted to use in the main window client area, and may also be created within dialog boxes. Control windows are clipped to the boundaries of their parent window according to the same rules as other child windows.

2.5 Icons

An **icon** is a small graphical image on the screen, used as a visual representation of an object such as an application or a window. While the use of icons is not restricted to object-oriented applications, the implementation of an icon-based user interface, where icons representing objects are directly manipulated on the screen by the user to achieve a desired result, is the essence of the object-oriented **workplace environment** implemented by the Workplace Shell under OS/2 Version 2.0. For more information see Chapter 6, "Installing and Supporting the Workplace Shell" and Chapter 8, "Workplace Shell Implementation."

Icons are designed using the Icon Editor program supplied with OS/2 Version 2.0. This utility offers the interactive design of icons, pointers and bitmaps by the user. The resulting objects are saved in files for subsequent use by applications.

The implementation of an icon depends on the associated application. There are three different possibilities:

1. For a full-screen protected-mode application (that is, those which do not utilize Presentation Manager facilities) or for an application which executes in a text window under Presentation Manager, an icon can be provided for use with the application. It must be in the same directory and with the same file name as the application executable module. For example, the icon for the application MBOOGLE.EXE would have the name MBOOGLE.ICO. When loading the application, the Presentation Manager automatically checks the directory for a corresponding icon file. If it exists, it will use it to represent the application. Otherwise a default icon will be used.
2. For Presentation Manager applications, icons are generally incorporated into the executable modules when the applications are created. See *OS/2 Version 2.0 - Volume 4: Application Development* for further information on creating icons and including them in Presentation Manager applications.
3. If there is no default icon available for the application, the user can associate an icon by using the "General" page in the Settings notebook. This dialog box allows the user to create a new one or to search for another available icon with the find command.

Data files generally have a default icon, but this can be changed in the same way as for an application.

2.6 Clipboard

The clipboard provides a temporary storage area for a piece of text, a bitmap or a metafile. It enables the user to move data within a single application or exchange data among applications. Typically, a user selects data in the application using the mouse or keyboard, then initiates a cut or copy operation on that selection. Using the paste command the user can insert this data into another place in the same application or in another application. All these operations are performed by applications.

Generally an application should first verify that no other applications are trying to retrieve or set clipboard data. Finally, when the application finishes its access to the clipboard data, it releases the clipboard so that other applications can use it.

These operations are described below:

Operation	Description
Cut	Deletes the selected data from the application and copies it to the clipboard. Any previous contents of the clipboard are destroyed.
Copy	Copies the selected data to the clipboard. The selection remains unchanged. Previous contents of the clipboard are destroyed. Before an application performs a **cut** or **copy** operation, it removes any previously stored data in the clipboard. Then it writes its data to the clipboard in as many standard formats as possible.

Paste Deletes any selected data from the application and replaces it with the contents of the clipboard. The contents of the clipboard are not changed. When the application performs a **paste** operation it verifies the format where the data are stored. If the clipboard contains one of the requested formats, such as text data, the application gets a pointer to a shareable memory object containing the text. For bitmap data or metafile, it gets a corresponding handle.

The clipboard is a small amount of system memory for user-driven data exchange. The clipboard only stores pointers to data. A set of API functions enables the application to move and exchange data. Figure 4 is an example of copying data from one application and pasting it to another, by way of the clipboard.

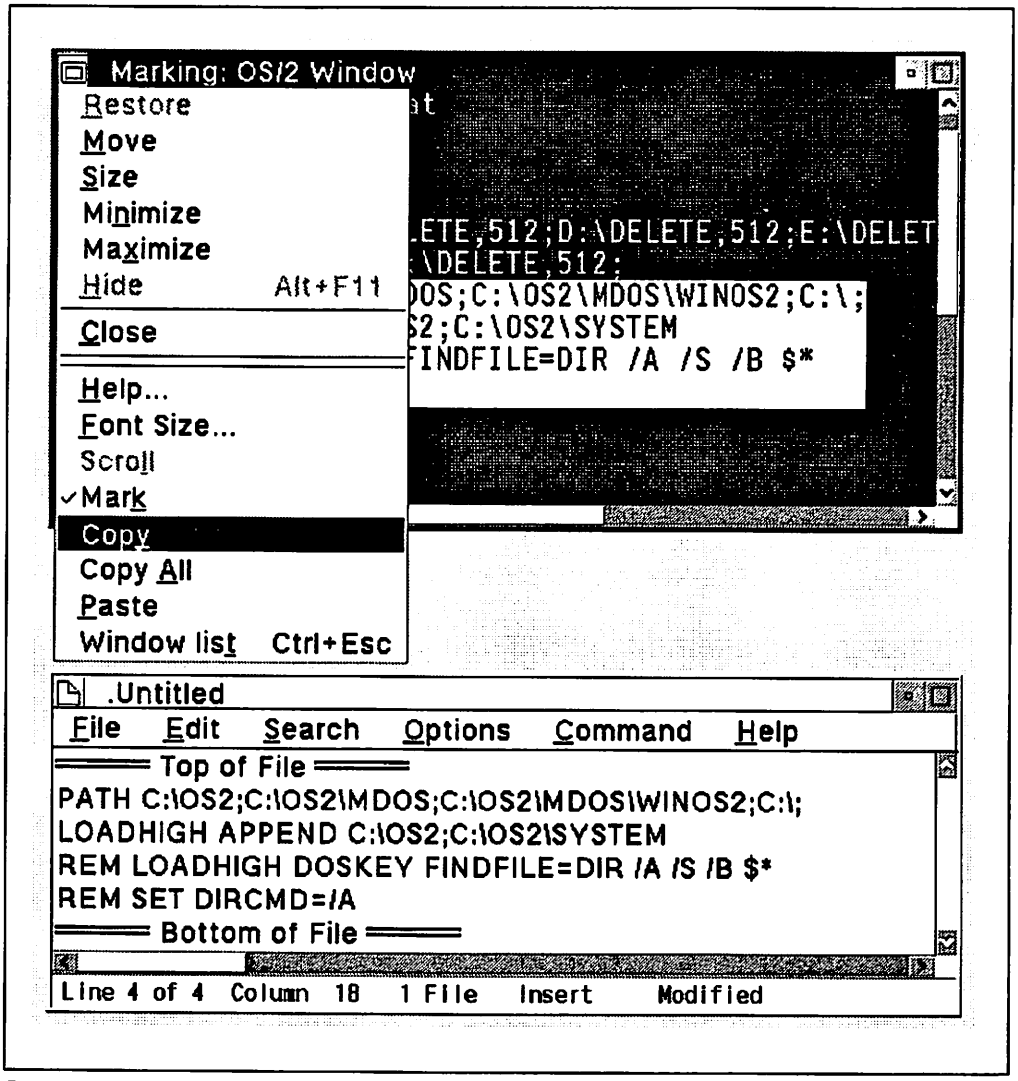

Figure 4. Clipboard Copy from an OS/2 Window into an OS/2 PM Editor

The data in the clipboard is maintained in memory only. Clipboard data is lost when the computer is turned off.

2.6.1 Shared Memory and the Clipboard

An application must store, in shared memory, text data that is destined for the clipboard. The application passes the clipboard a pointer, which the clipboard uses to access the shared memory object. To pass a bit map or metafile to the clipboard, an application passes the clipboard a bit map or metafile handle. The clipboard functions make the bit map or metafile *shareable*.

2.7 Summary

The Presentation Manager user interface provides the capability for the user to interact with applications in a consistent, intuitive manner. Presentation Manager components conform to the guidelines laid down in the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design*.

The Presentation Manager user interface makes extensive use of windows and a series of defined user interface constructs which appear and behave in a consistent manner, and which may be used by applications to interact with the user. Special-purpose windows such as dialog boxes and message boxes are also supported by the interface for use in specific circumstances, and their appearance and behavior is also defined and regulated by CUA guidelines. The use of such predefined constructs by applications allows a high level of consistency between applications in terms of their interaction with the user, thus simplifying the level of training required to use those applications.

The event-driven, object-action style of the user interface implemented by Presentation Manager is highly intuitive. This fact, combined with the high level of consistency between applications, encourages learning by exploration. In turn, this helps reduce the need for formal training and enables users to become more productive with new applications in a shorter period of time.

Chapter 3. New Presentation Manager Features

The component of the operating system which is responsible for interaction with the end user is known as the user shell. The PM Shell in OS/2 V1.3 has been replaced in OS/2 Version 2.0 by an enhanced, object-oriented user shell known as the *Workplace Shell*, which implements the 1991 CUA Workplace Environment.

The Workplace Shell allows users to become more task-oriented by simplifying the user interface and reducing the amount of system-specific knowledge required to perform work tasks. The high degree of consistency in the operating system also reduces the amount of user education needed to operate the system.

The shell has been redesigned for OS/2 Version 2.0 to give the user a single interface to manage multiple types of objects, including devices (printer and drives), files, and programs. Each defined printer or attached drive is a separate icon. These objects are arranged at will on the desktop or in specific shell windows. The user interacts with the objects using a well-defined drag and drop environment and is able to manipulate files without needing to be concerned about the file directory hierarchy. However, if the user wishes, multiple directory trees are accessible simultaneously. Views, selection techniques, and actions are consistent throughout the shell.

This chapter reviews the changes to Presentation Manager. Some of the information has been abstracted from articles written by the developers in the *IBM Personal System Developer, Winter 1992.*

3.1 New Window Classes

A number of new control window classes have been added to Presentation Manager under OS/2 Version 2.0. These controls aid in the implementation of the Workplace Shell and provide enhanced function and flexibility for user dialogs in Presentation Manager applications.

This chapter describes these new controls and their interaction with the end user. More information on using these controls in applications can be found in *OS/2 Version 2.0 - Volume 4: Application Development.* and the *OS/2 Version 2.0 Programmers Guide, Volume II.*

3.1.1 Container

Folders are extensively used within the Workplace Shell to logically group objects (represented by their icons) on the desktop. A folder consists of a standard window frame plus a container control which occupies the whole of the client area. A folder can contain other folders, objects representing work items such as reports, or physical items such as printers. The objects displayed by a container control are determined by the application.

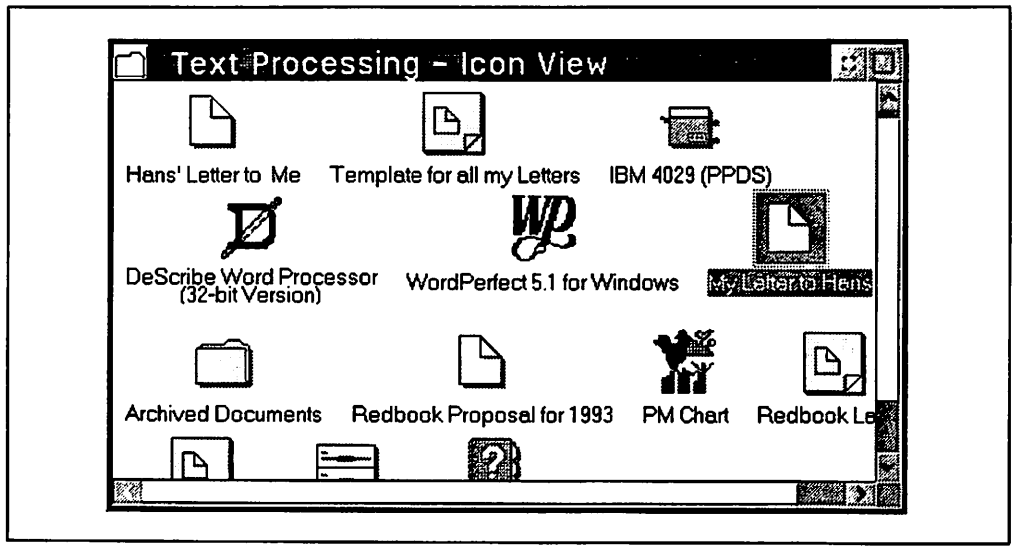

Figure 5. Container Control Used in Folder Window

Various types of objects may be displayed within a container; these include objects representing printers and other physical devices, as well as those representing items such as files or documents. Information about these objects can be presented in a variety of views. Each view describes the objects in a different format, some giving different and/or additional information. The container supports the following views of its data:

Icon view This view displays either icons or bitmaps, with accompanying text beneath them, to represent the items in the container. This is the default view for a container.

The user may group the objects as he chooses. He can overlap them, in a "messy desktop" fashion, or can have them automatically positioned. The default icon view is not "gridded"; that is, it has free-form characteristics so data can be placed in various relative positions and still have meaning. However, if a gridded display format is preferred, the objects can also be arranged in rows from left to right and from top to bottom.

The name view This displays either icons or bitmaps with accompanying text to the right of each icon or bitmap, representing the items in the container. Items are arranged in a single column. A variant of the name view is the **flowed name view**, where items are arranged in multiple columns.

Text view This view is similar to the Presentation Manager list box in OS/2. Text strings are displayed in columns. This view also offers the option of flowing the text into multiple columns to fit the windows when it is sized.

Details view This view allows the user to display detailed information about items in an unlimited number of scrollable columns. The data within each column may be icons, bitmaps, text strings, or national language support (NLS) formatted date or time strings. An optional splitbar may be used to divide the display area into two windows which scroll independently horizontally and dependently vertically.

Tree view This presents data in a hierarchical format, similar to the directory layout used in the OS/2 Version 1.3 File Manager. The leftmost items displayed in the tree view are at the root level. Root level items can contain other items called children. These can be displayed in the tree view. If the children are not displayed, the parent item can be expanded to display them as a new branch in the tree view.

Each instance of text in the container can consist of unlimited lines with unlimited characters in each line. In addition, the container control does not limit the number of objects within a container. Objects are automatically arranged within the client area of the container.

Direct manipulation is supported in all views of the container control. This allows the user to drag container items within a current window or from one window to another.

Containers and their use by applications are further explored in *OS/2 Version 2.0 - Volume 4: Application Development*.

3.1.1.1 User Interaction

The end user can alter any text field in the container, including the container title, column headings in details view and all container items, through direct editing. Text strings can also be individually set to a "read-only" state.

Objects within a container may be selected through marquee, touch swipe, range swipe, and first letter selection techniques. The container control supports single, extended, and multiple selection types; these are described in the *IBM Systems Application Architecture CUA Advanced Interface Design Reference*.

3.1.2 Notebook

The notebook control is used to provide an easy and intuitive method for a user to navigate through a complex dialog, where multiple related dialog boxes are displayed. The notebook control allows the programmer to assemble a collection of dialog boxes which relate to a single topic. It is designed to visually resemble a bound notebook with multiple pages.

The notebook control provides an easy-to-use user interface component that is consistent across multiple products. In this way, it helps products to conform to the Common User Access user interface guidelines.

The notebook used to provide the desktop setting is shown in Figure 6 on page 24.

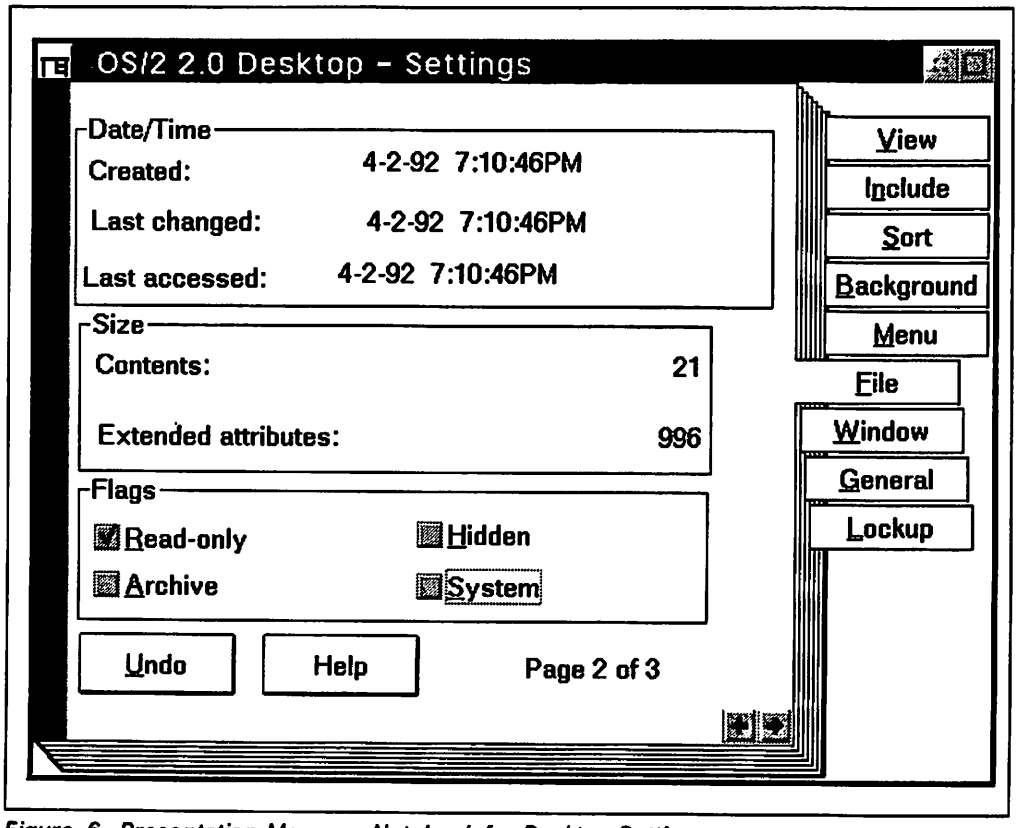

Figure 6. Presentation Manager Notebook for Desktop Setting

Data in the notebook is presented on pages bound together on one edge. Pages appear recessed on two edges of the book, thus providing a three-dimensional appearance.

The notebook control supports the use of both a pointing device, such as a mouse, and the keyboard for displaying notebook pages and tabs, and for moving the selection cursor from the notebook tabs to the top page. The end user can turn from page to page or may go quickly from one tab page to another. The following navigation components are provided:

Section dividers Across from the notebook binding are the major section dividers (called *major tabs*). The section dividers provide a means of organizing the data within the book. Each section within the notebook may contain single or multiple pages. The notebook provides a method for the end user to turn the pages within a section and also to skip from one section to another easily. Just as dividers provide an indication of where the user is within the book, methods are supported to indicate where the user is within a section.

Page buttons In the bottom right corner of the notebook in Figure 6 are the *page buttons*. These buttons are used to view one page of the notebook at a time. Selecting the *forward page button* (the right-pointing arrow) causes the next page to be displayed, while selecting the *backward page button* (the left-pointing arrow) causes the previous page to be displayed.

The visible area of the notebook is the *top page*. The application uses this top page to display information and facilitate user interaction. The top page may

contain application-created windows or dialogs. Only one page is visible at any time. The notebook handles the hiding and showing of the topmost window or dialog when pages are turned.

If all the tabs currently inserted cannot be displayed, *scroll arrows* are provided to scroll the tabs forward and backward. Figure 7 shows a notebook used by the Master Help Index:

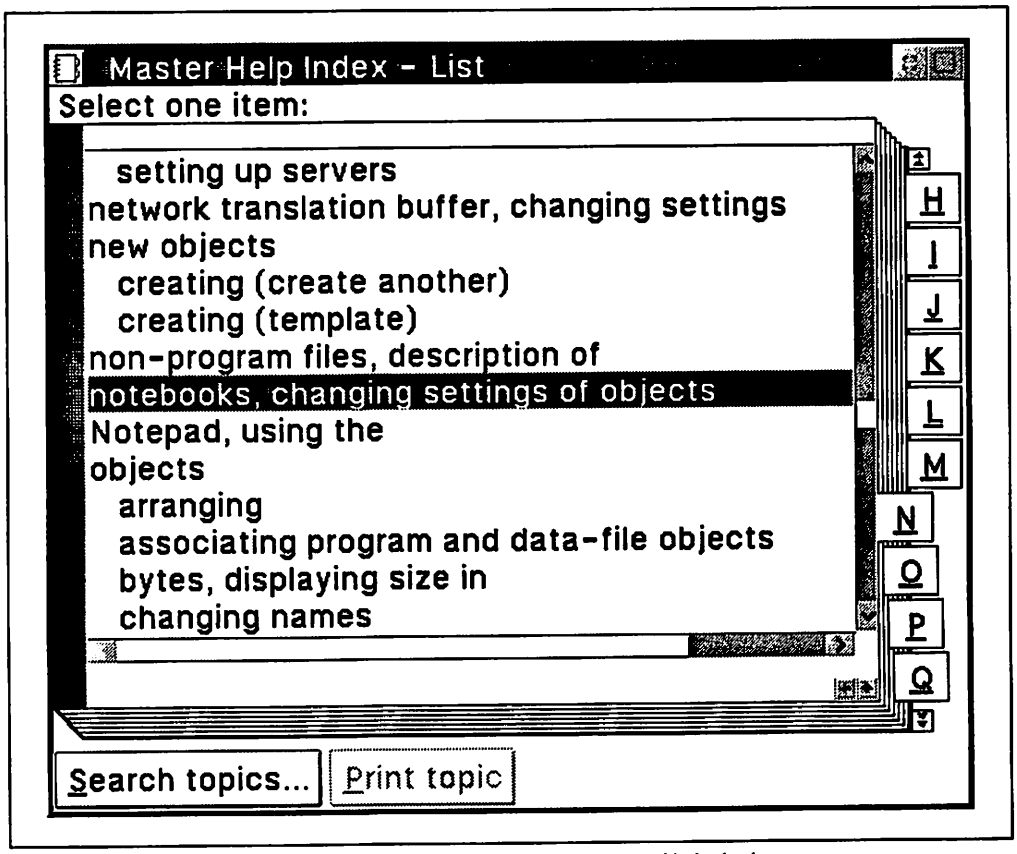

Figure 7. Presentation Manager Notebook used in Master Help Index

The notebook control is comprised of various regions which may be dynamically resized. Whenever the notebook window is resized or any of the notebook visual regions are resized the notebook dynamically recalculates the sizes of all affected regions for future display.

3.1.3 Slider

The slider control supports the setting of approximate values and properties in an analog rather than digital form. It is designed to be used where settings are intuitively expressed in analog or relative form, rather than exact numeric values.

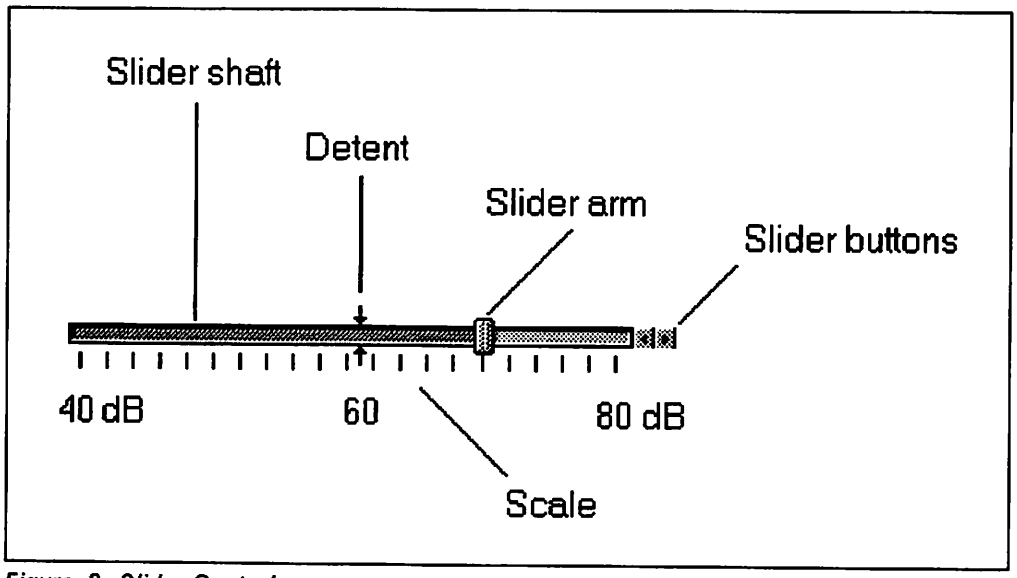

Figure 8. Slider Control

The slider control consists of a *slider shaft*, within which is a *slider arm*. The arm is used to change the setting of the slider control. The arm can be moved by the user by dragging it with the mouse or by using the cursor arrow keys on the keyboard.

An application may also change the setting of a slider control from within the application, independently of user action. The application creating the slider control must specify the range of available values or increments for the slider and may also specify the spacing for items in the rulers.

In previous versions of OS/2, the scroll bar control was often used to provide the same effect as the slider; for example, setting colors in the control panel. The slider control is better than the scroll bar in such cases, since it is more flexible and typically easier to control within an application. Providing a slider control also means that the scroll bar can be used only for its intended purpose of scrolling information within a window, which results in a more consistent user interface.

3.1.4 Value Set

The value set control is similar to the existing radio button control since its purpose is to allow a user to select an item from an existing set. However, unlike radio buttons, the value set provides a graphical set of selectable items. Suppose an application, like the one shown in Figure 9 on page 27, provides the ability to select a drawing tool. Using the value set, the application can provide a graphical image of the different tools so that the user can see all the available choices when he makes his selection.

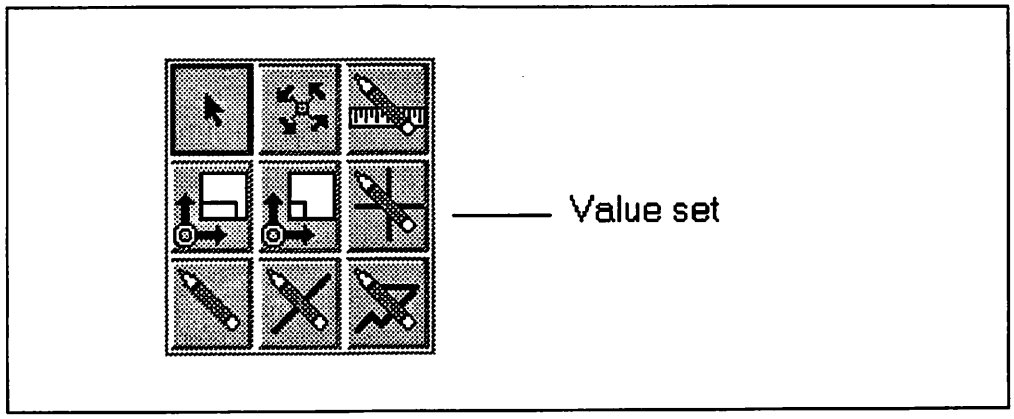

Figure 9. Value Set Control

The value set control offers great flexibility to the interface designer because it can be extended as the application grows. The system drag protocol is supported by the value set control. This means that users could, for example, select a color from a value set and then drag that color to the target item. For example, he could drag the color to the client area of a window to change that window's background color.

The items within a value set control may include:

- Bitmaps
- Icons
- Text strings
- Colors.

Items of different types may be mixed in the same value set control.

While a value set control may be used to display textual or numeric data, it is recommended that a radio button be used for this purpose, and that the use of the value set control be confined to the display of graphical data.

3.1.5 Progress Indicator

The progress indicator displays the progress of a long-running command to the user. It is typically used for operations such as uploading or downloading files to or from a remote system, backing up a fixed disk etc. The progress indicator provides an easy way for different applications to display such progress to the end user in a consistent manner.

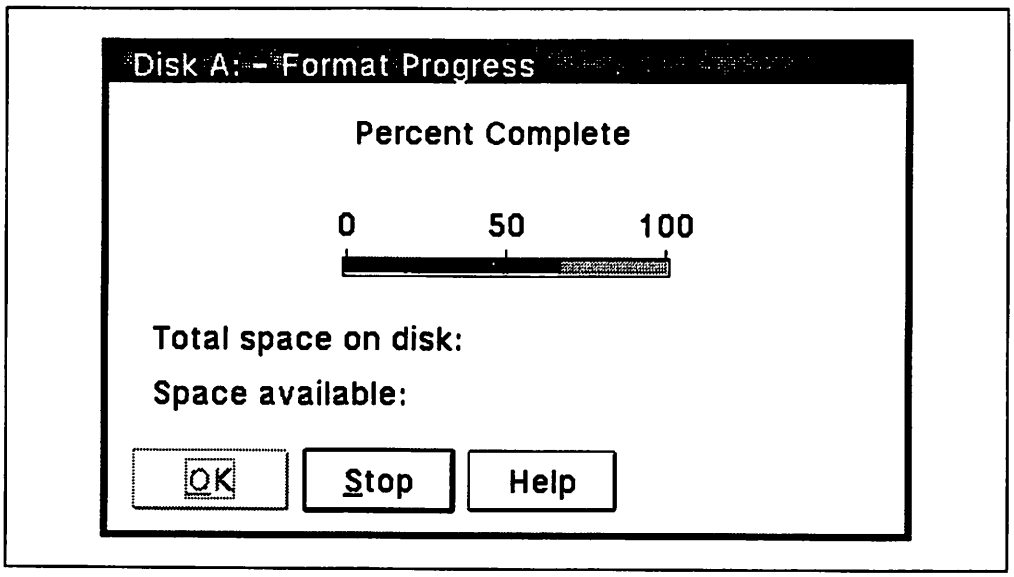

Figure 10. Progress Indicator Control

The progress indicator includes push buttons which allow the user to control the operation in progress. The user may pause or resume the operation; the operation may be cancelled when in the paused state. The progress indicator control passes messages to the application when the user selects any push button.

3.2 Standard Dialogs

OS/2 V2.0 provides a number of standard, CUA-conforming dialogs which may be used by applications to perform common operations such as file handling and font selection. The provision of these dialogs within the operating system avoids the need for applications to individually develop dialog boxes and procedures for these functions. This delivers a high degree of consistency between similar operations in different applications.

Both forms of the standard file dialog are accessed by an application using the **WinFileDlg()** function. This function is explained further in *OS/2 Version 2.0 - Volume 4: Application Development*.

3.2.1 File Dialogs

There are two standard dialogs provided for file manipulation: the *Open* dialog and the *Save as* dialog. Each provides similar function, but has slightly different behavior due to the different function being performed with the dialog. The use of these dialogs enables application developers to provide these functions in a consistent manner, without having to code such function into their application programs.

The standard "Open" dialog box, used to select and open directories and files from within an application, is shown in Figure 11 on page 29.

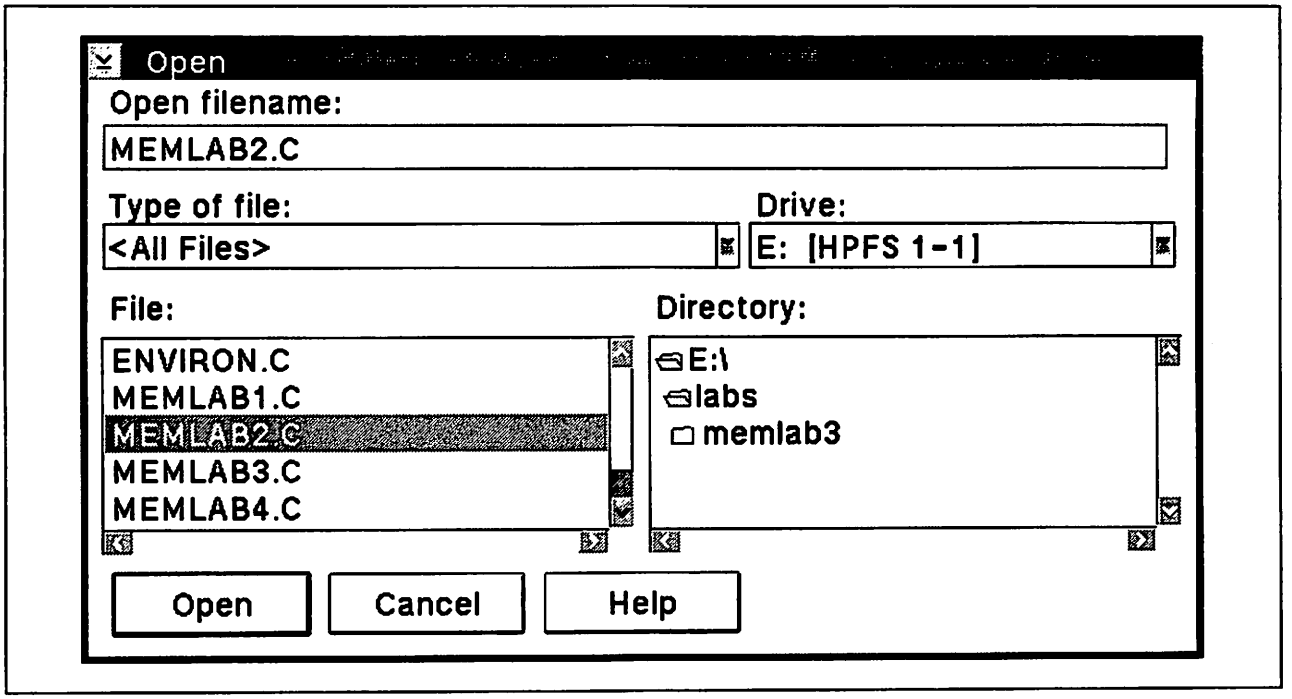

Figure 11. Standard File Dialog for "Open"

The file dialogs are required by application-oriented programs wishing to conform to the 1991 CUA guidelines. By standardizing the commonly performed file manipulation tasks across different programs, they help to eliminate minor inconsistencies in dialog operation, which results in fewer user errors.

3.2.2 Font Dialog

The Workplace Shell also provides a standard font dialog. The font dialog is available to any application that wants to let a user view and select fonts. The basic functions of the font dialog provided include the ability to select from:

1. A list of font family facenames installed on the system
2. Styles for each font
3. Available sizes for each font
4. Emphasis styles available for each font.

The font family facename is defined as the name of the typeface, for example, Courier, Times New Roman, and Helvetica. Type styles include normal, bold, italic, and bold italic. Size is defined as the point size, or vertical measurement of the type. Font emphasis styles include outline, underline, and strikeout.

Figure 12 on page 30 shows the appearance of the standard font dialog.

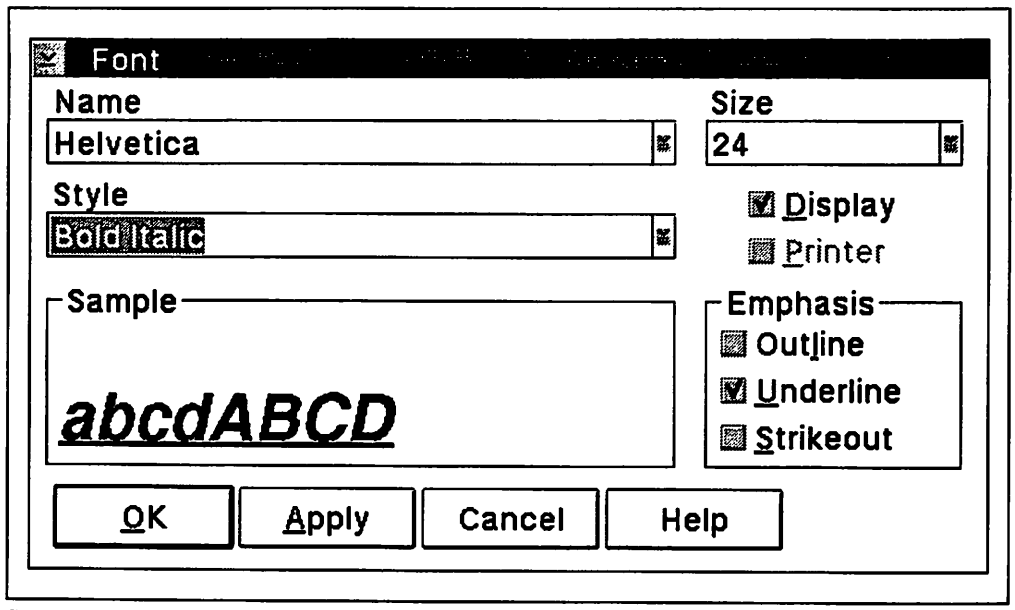

Figure 12. Standard Font Dialog

The font dialog also has a viewing area that is updated dynamically when the user makes selections of fonts, styles, etc. This viewing area lets the user preview any font prior to applying it.

3.3 Information Presentation Facility

OS/2 Version 2.0 provides a number of enhancements to IPF in addition to the capabilities available under OS/2 Version 1.3. This section briefly describes these enhancements; more complete information may be found in the *IBM OS/2 Version 2.0 Information Presentation Reference*.

The **Information Presentation Facility (IPF)** allows application developers to provide online, context-sensitive help information for Presentation Manager applications. IPF also allows online manuals and documentation to be built and viewed independently of applications. Automatic table of contents, searching and printing facilities are provided by IPF.

The *hypertext links* have been extended to include bitmaps and metafiles, in addition to text phrases. These hypergraphic links provide enhanced flexibility for the display of online documentation, particularly procedure manuals and tutorials. Hypergraphic links may be used to display additional information, send a message to an application, or start a new process in a similar manner to hypertext links.

Under OS/2 Version 2.0, links may be made between panels which reside in different source files; these files are concatenated and the resulting information may be viewed as a single entity. Both hypertext and hypergraphic links are supported in this manner. This enables larger amounts of information to be viewed, and allows the separation of volatile information into separate files for easier update.

The split screen support provided by IPF under OS/2 Version 2.0 allows a window to be regarded as **multiple viewports**, each of which may be manipulated separately by the author and the user.

Under OS/2 Version 2.0, all windows are composed of one or more viewports. The default viewport is simply the entire window. IPF Version 2.0 allows **secondary viewports** to be opened, to display related information which may be scrolled and manipulated separately from the parent information.

IPF provides two different types of viewport within a window:

- IPF-controlled (IC) viewports are created, controlled and manipulated by IPF in a similar way to normal IPF windows.
- Application-controlled (AC) viewports are controlled by the application, and are typically used for the display of specialized information. For example, an AC viewport might be used to display animation or full-motion video.

These different types of viewports may be mixed within the same parent window.

Dynamic data formatting allows applications to place customized information in help windows or online documents at run time. For example, this facility could be used in a help window to display the data from a current transaction to the user, along with the required steps to complete that transaction. The help information is thereby made more relevant to the immediate task.

IPF allows items within help windows to be defined as **hypertext**. When the user selects such items, actions may be triggered (for example, opening another help window to display more detailed explanatory text, posting a message back to the application's window procedure to invoke an application event, or starting another process under OS/2).

Under OS/2 Version 2.0, IPF supports the use of multiple fonts within help windows or online documentation, as well as multiple character sizes for each font. By default, all help windows and online documentation windows appear in the system font. The Courier, Helvetica (Helv), and Times Roman (Tms Rmn) fonts are available in all display adapters supported by Presentation Manager. Where the specified size is not available for the required font, the closest match is used.

3.4 Summary

The new control window classes provided under Presentation Manager in OS/2 Version 2.0 help in the implementation of the Workplace Shell and enable application developers to create applications which more easily integrate with the 1991 SAA CUA Workplace Environment.

Icons and containers are fundamental to the Workplace Shell, and their inclusion as control window classes provides the means for consistent behavior of such objects between all applications on the desktop. The notebook control allows the display and navigation of complex user dialogs in an organized manner, within the context of the Workplace Shell.

The slider, value set, and spin button control window classes provide additional flexibility for end-user dialogs, enabling applications to present properties to the user in a more meaningful and less confusing manner than was the case in previous versions of OS/2.

The progress indicator control window class allows applications to display progress information for long-running operations, in a consistent manner. The

implementation of this capability as a control window eliminates the need for application developers to explicitly code such function into their programs.

Functional enhancements have been made to IPF under OS/2 Version 2.0, resulting in greater functionality and flexibility in information panels produced for help files or online documentation. This enhancements includes support for multiple viewports, support for multiple fonts and character sizes.

Chapter 4. Workplace Shell Components

OS/2 Version 2.0 provides an improved user shell, that is, the component of the operating system which is responsible for the appearance and behavior of the user interface. This shell is based upon the 1991 IBM Systems Application Architecture (SAA) Common User Access (CUA) Workplace Environment, and is known as the **Workplace Shell**.

One of the historical drawbacks of the OS/2 environment has been the power and therefore the complexity of the operating system. Much of this complexity has, in the past, been allowed to transfer itself to the user interface, thus often requiring a large amount of user effort to complete a task.

The objective of the Workplace Shell is to simplify and facilitate the performance of work tasks by an end user through the use of graphics and the direct manipulation of icons on the screen. The Workplace Shell aims to insulate the end user from the complexity of the OS/2 operating system, requiring less effort to complete a particular task, thereby reducing the level of knowledge and experience required for the user to successfully manipulate the system.

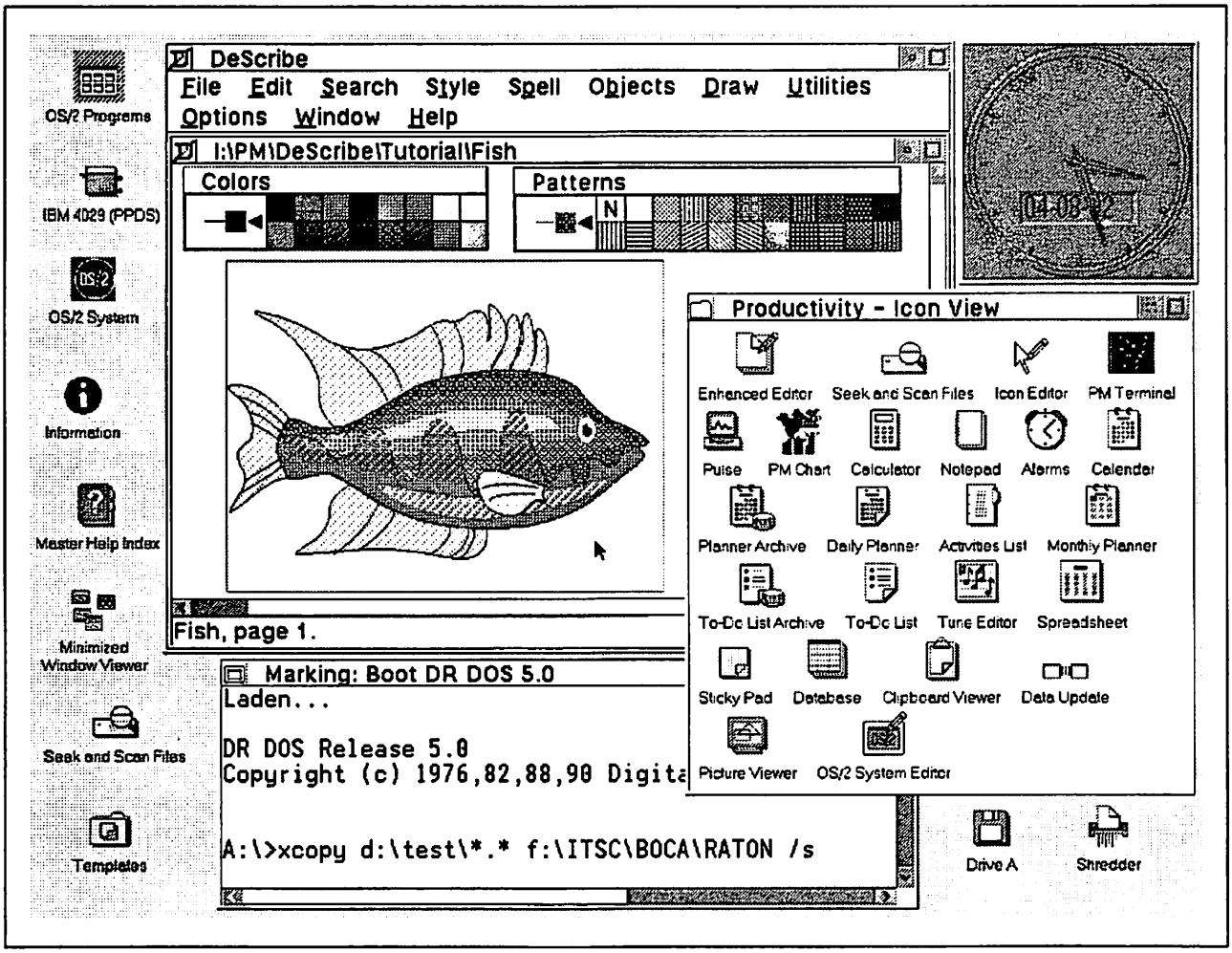

Figure 13. Workplace Shell Desktop Appearance

In addition, the Workplace Shell provides a more consistent user interface by implementing the additional control window classes and standard CUA functions

provided under OS/2 V2.0, thereby easing the task of working with multiple applications concurrently, and of learning new applications.

4.1 CUA and the Workplace Shell

In previous versions of OS/2, the user shell was designed to conform to the 1989 SAA CUA Graphical Model. The Workplace Shell of OS/2 Version 2.0 now reflects the 1991 SAA CUA Workplace Environment.

In May 1989 (as part of the OfficeVision* announcement) IBM announced an extended role for the programmable workstation in the SAA environments. Part of this announcement, known as CUA 89, introduced the Workplace extension to CUA's graphical model. This workplace environment defined a more object-oriented approach to interaction with the system through direct manipulation of objects.

In September 1991 IBM announced extensions to the SAA architecture. Included in these extensions was CUA 91, an orderly growth from CUA 89. CUA 91 enhances the object-based user interface defined in CUA 89. Rather than interacting with applications, users interact with **objects** which represent the inputs and outputs of their jobs. The Workplace Environment is based upon the metaphor of the user's working environment, with objects such as forms, letters, an address book, printers etc., represented on an electronic **desktop** using graphical symbols known as **icons.**

CUA 91 has an increased emphasis on **direct manipulation;** that is, the manipulation of workplace objects through their representative icons on the desktop. This allows the user to concentrate more on the task at hand, and less on the mechanism that must be employed. The user is presented with a number of icons, each of which is a pictorial representation of the real underlying object which in fact controls the data.

CUA 91 also introduces a number of new controls. Perhaps the most noticeable of these is the **notebook control.** This control allows an application to present a multi-page dialog (for example, a pad of blank forms, a log or register, or a set of reference notes). The notebook control provides a more meaningful way of providing an electronic metaphor of complex objects than a series of dialog boxes or other secondary windows.

Extensive documentation for CUA 91 exists in the form of the *IBM Systems Application Architecture CUA Advanced Guide to User Interface Design* and the *IBM Systems Application Architecture CUA Advanced Interface Design Reference.* However, the best way to appreciate the spirit behind CUA 91 is to view the demonstration "The CUA Vision - Bringing the Future into Focus." This demonstration is based on a version of CUA beyond CUA 91. It does, however, bring home very vividly the essence of CUA 91: pervasive interaction with workplace objects (represented by icons and other graphical controls) through direct selection and manipulation. As will be seen in this demonstration, the user is not aware of applications as such. Business processes are completed as a result of a natural and meaningful interaction with objects.

4.1.1 Icons

Studies have shown that the human eye can identify and distinguish graphical information more easily and effectively than textual information, and that visual appeal is an important motivating factor in the workplace. The Workplace Environment makes extensive use of icons to represent objects related to user's work tasks. Icons provide a very effective means of identifying a required object, particularly where many possibilities are displayed to the user. Still it is advisable not to put too many icons into any workplace. Depending on the user, more than 10 to 20 icons in a given area may be confusing. A graphical environment encourages an orderly arrangement.

The icon is a two-dimensional depiction of the object. It should resemble the physical object it represents but it should also resemble what the user *recognizes* as an object of that type. For example, if the user expects a printer to look like the one in Figure 13 on page 33, then an accurate portrayal of an IBM 4029 Page Printer is not going to be easy for that user to recognize and find; the user will have to read the title below the icons to locate a printer.

A detailed discussion of these issues can be found in the *Icon Reference Book, SC34-4348.*

The icon editor is part of OS/2. The user can change the icon of any object from its *Settings* view. The "General" page shows the current icon and offers the option to edit that icon. When "Edit" is selected, the Icon Editor is invoked.

4.2 An Introduction to the WPS

The Workplace Shell is radically different from previous versions of OS/2. At first, it can make users of former versions of OS/2 feel insecure and it takes some time to really appreciate the many advantages of the new interface.

OS/2 comes with a tutorial program that can help both experienced and new users learn how to use the elements of the Workplace Shell. This is started automatically once after the installation but can be called later at any time.

The shell consists of an *electronic desktop*, on which are placed various *icons*. These icons represent objects such as folders, data files, program references and physical devices. The icon tells the user about the object it represents. Some icons look like folders, while others look like printers. Others may look like a book, a car or an order form.

It is important to understand that the symbols on the screen are just that. A symbol (or icon) may represent a real object or it may be a **shadow copy** of a real object. The icon provides no differentiation between an object and its shadow copy, although description text of the shadow is gray.

Deleting a shadow copy does not delete the real object. Therefore many large customers may prefer that their users work with shadow copies to prevent them accidentally deleting the real objects.

4.2.1 The Desktop

4.2.1.1 Background

The background is the total space on the desktop. It is a special form of a container but follows all basic rules for a folder. It has a context menu which is activated using mouse button 2. Not all of the background may be visible.

4.2.1.2 Size

The desktop allows you to organize (or disorganize) documents and files the way you would on a real desktop. It cannot be smaller than the physical screen. This desktop concept helps the user to organize things and thereby focus on specific groups of objects.

4.2.1.3 Enlarging the Desktop

The size of the desktop starts out as large as the screen. When more objects are put on the desktop than the basic size can hold the desktop will be extended automatically. Scroll bars are added to the desktop if objects are put beyond the edges of the screen.

4.2.1.4 Changing the Background

The Workplace Shell allows the user to specify a color or a bitmap to use for the desktop background. Individual colors or bitmaps may also be chosen for any folder in the system.

The desktop background is specified from the *Background* page of the desktop *Settings* view. The user may specify one of the following:

- An **image** uses a single bitmap to cover the entire desktop. The user has the option to display:
  - A **normal** image, where the bitmap is displayed at its normal size, centered on the desktop or folder
  - A **tiled** image, where the bitmap is repeated many times to cover the entire desktop or folder
  - A **scaled** image, where the bitmap is stretched to cover the entire desktop or folder.
- A **color** simply sets the entire desktop background to the required color.

A bitmap may be designed using various graphics packages, or may be standard bitmap-format files obtained from other sources. A number of public bulletin board services contain repositories of such bitmaps.

4.2.2 Objects On The Standard Desktop

In previous versions of OS/2 the icons represented applications, but in the Workplace Shell they may represent both applications and objects. The appearance of an object can be either "general" or "personal."

The general appearance is usually inherited from the class to which the object belongs. There is no way, short of programming, to change the class icons, such as folders and data files, which are shipped with the Workplace Shell. However, any icon can be individually changed, or "personalized," by the user. Icons which have been changed are stored in the *Extended Attributes* of the object.

When the user starts OS/2 V2.0, the following objects are visible on the desktop:

Printer This is the default printer attached to LPT1.

Shredder This deletes any object (icon) dropped on it.

Minimized Window Viewer
All minimized window's icons will be put into this folder for easy access.

OS/2 System This folder contains the objects which allow the user to tailor certain properties of the operating system, such as mouse characteristics, screen colors and fonts.

Start Here This provides the user with a quick overview of the system. Details are found in the folder "Information."

Information Several objects containing information, like Tutorial and Command Reference, are held in this folder.

Master index This object provides an index to all on-line documentation.

Templates The contents of this folder allow the user to create new files, folders, programs and various other objects.

Drive A: This container contains a directory listing of the diskette in drive A:.

Most data objects are presented to the user in folders, which act as containers to logically group sets of objects. The objects within a folder may be rearranged by the user, and objects may be moved or duplicated between folders, allowing users to customize the desktop to suit their own working style. A user may place commonly accessed objects onto the desktop itself to avoid having to open a folder to access the required object. A typical example of a desktop is shown in Figure 15 on page 39.

4.2.3 Objects And Views

An object may have several visual representations, depending upon the nature of the user's interaction with the object. These representations (or **views**) can be changed by the user in order to show a different view of the objects. Figure 14 on page 38 shows two views of the objects on the desktop.

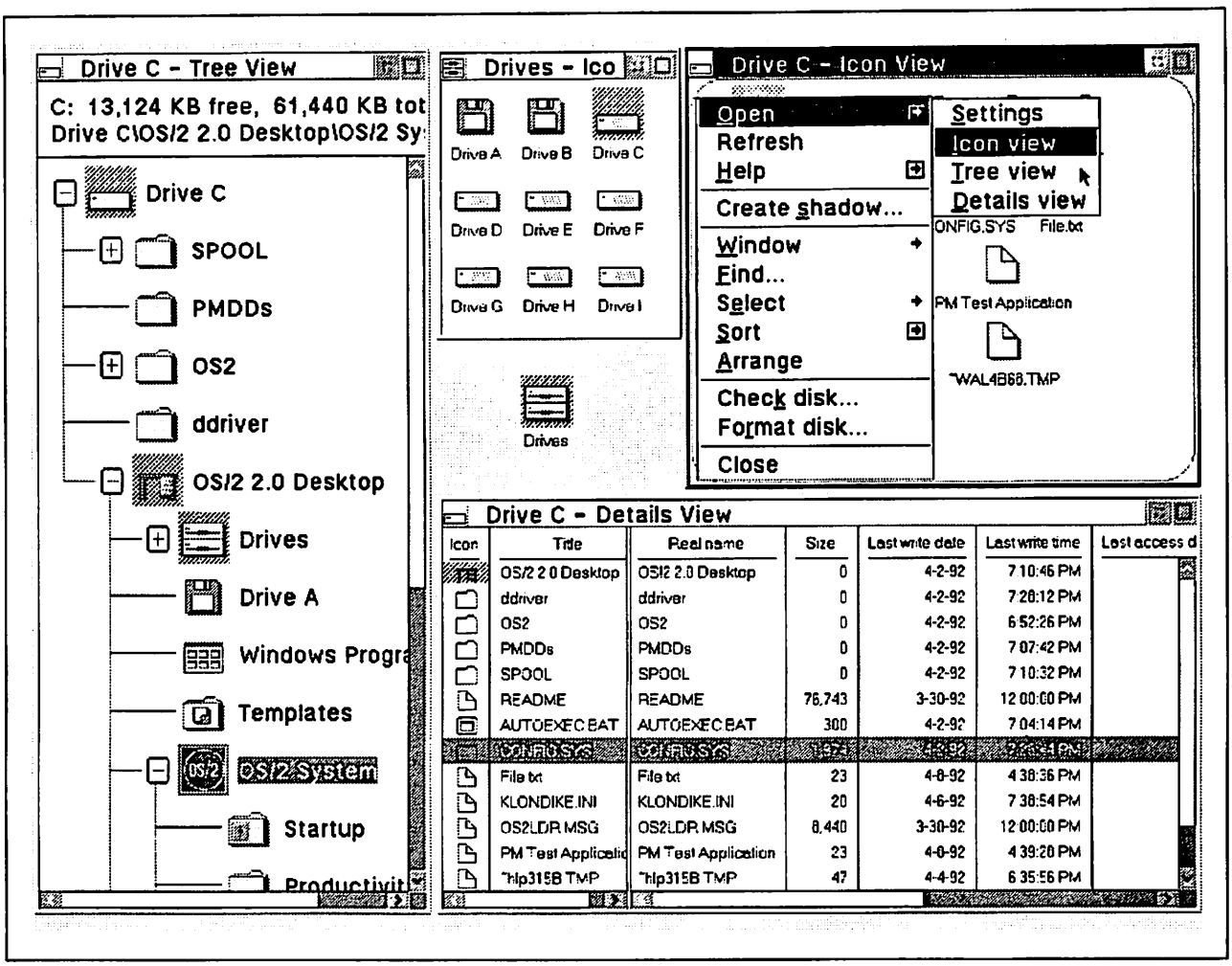

Figure 14. Different Views of the Same Objects

4.2.4 Customizing The Workplace Shell Objects

The user may want to place the most frequently used objects in a way that is most comfortable for him. Furthermore, the user may want to add or change colors. Some of these requirements are supported on a global scale, others on an individual object basis.

There are a few settings that are general to all applications. They can be found within the *OS2 System* folder under *System Setup*.

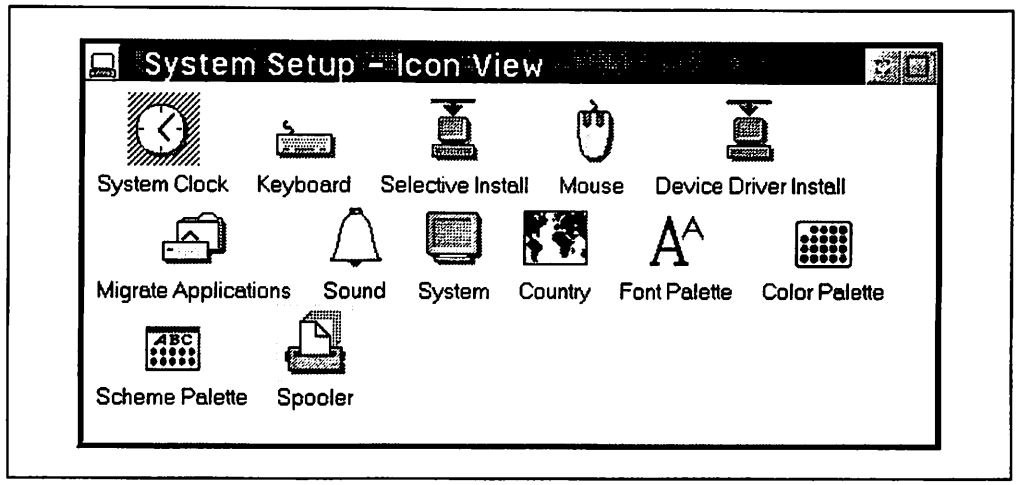

Figure 15. System Setup

4.2.4.1 Colors

A Color Palette and a Scheme Palette can be set for later use in modifying the appearance of windows. Included in the Color Palette is the setting for the border width. The information is stored in OS2.INI and can be changed easily for the whole system.

4.2.4.2 Font Palette

The palette of fonts available in the system can be changed by adding or deleting fonts. Certain display attributes like underlining and bold display can be set from here. Items can be dragged from the palette and dropped on the desired object(s).

4.2.4.3 Mouse

Quite a number of settings are available to control the behavior of the mouse buttons. Changing them, however, may cause trouble if more than one person is using the system.

4.2.4.4 Keyboard

Timing, special needs (see Figure 16 on page 40) like sticky keys, and the mapping of the function keys are defined in this selection.

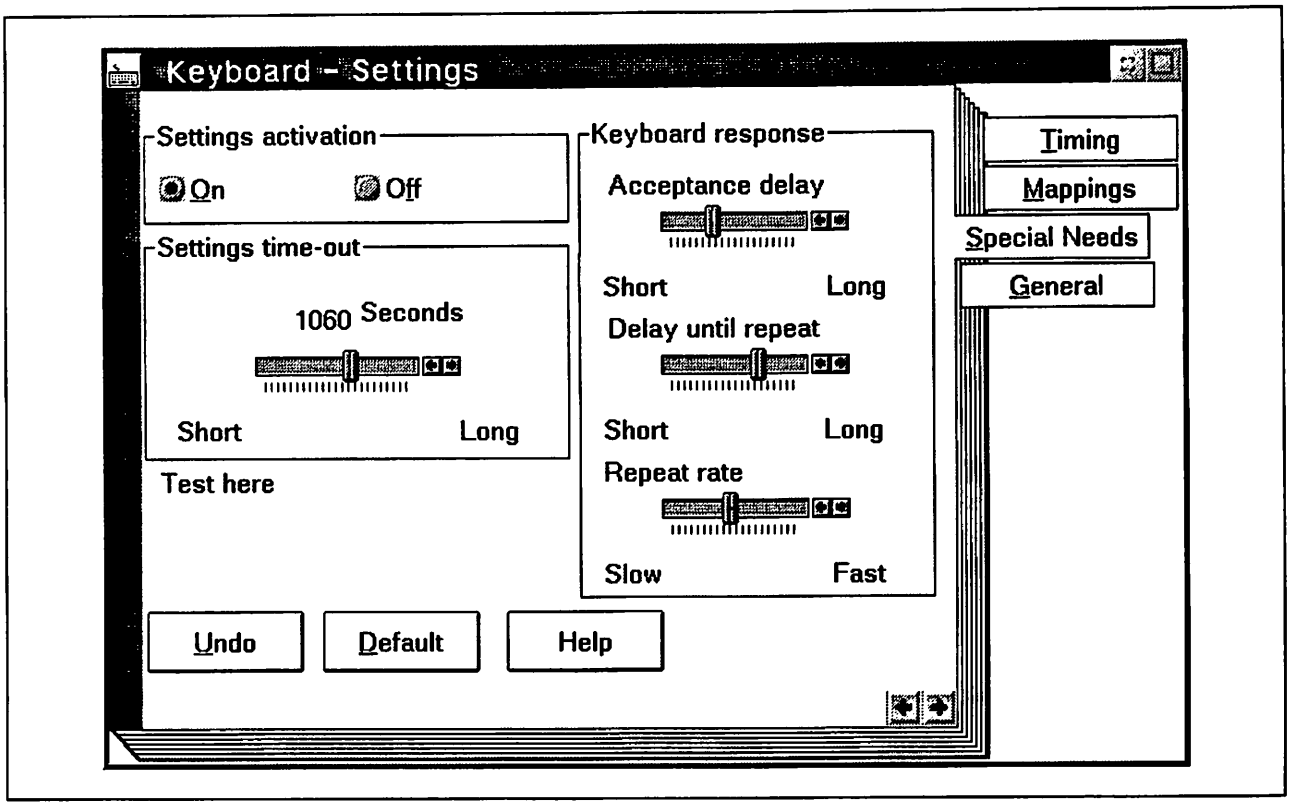

Figure 16. Keyboard Settings Notebook

4.2.4.5 Sound

This variable can be used by an application to find out if beeps are generally wanted. It does not stop an application from generating sound output.

4.2.4.6 Clock

The system clock can be displayed and various settings can be altered.

4.2.4.7 Country

The installation determines the settings like language group, keyboard, date and time formats. They can be changed at any time.

4.2.5 Arranging Folders and Objects According to Tasks

The user is free to create folders for all kinds of purposes. The major reason for folders is the organization of tasks. One folder may have all the objects in it which are needed for word processing, another one may be related to book-keeping, and so on. The user opens only the folder that is needed at the time and therefore the desktop can be kept tidy.

The user can "personalize" the folder for each task. For example, he can change the icon to help him find the folder more quickly. He can also make the folder into a work area; this tells the WPS to close any objects within the folder when the folder is closed and reopen them automatically when the folder is next opened.

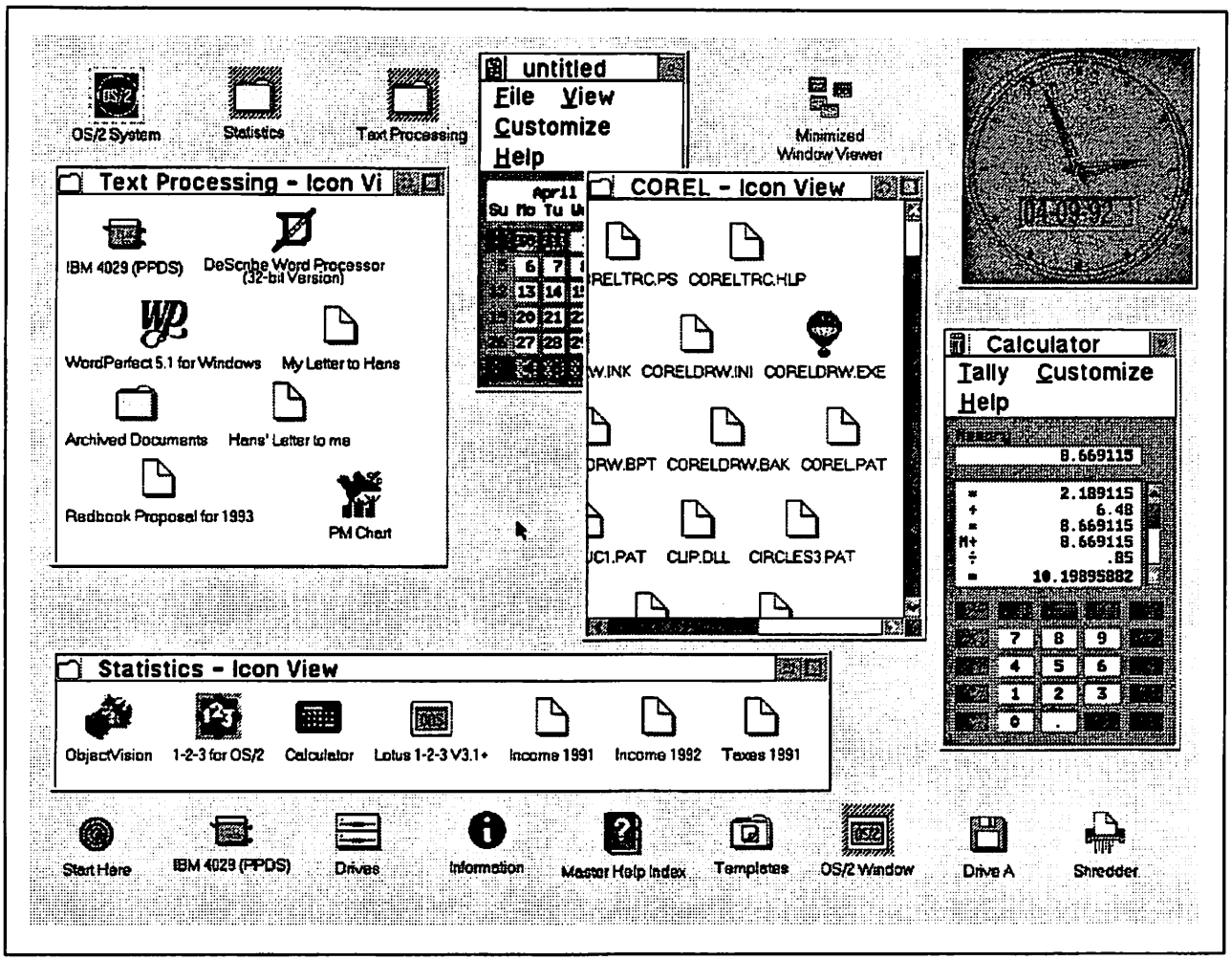

Figure 17. Workplace with Objects

4.3 Workplace Shell Objects

There are three main classes of objects defined by CUA 91 and implemented in the Workplace Shell:

- Device objects (such as printers and shredders)
- Container objects (such as folders and work areas)
- Data objects (such as files).

Each object is represented on the desktop by an icon.

Objects typically have a default action which is performed on other objects which are dropped on the object. For example, a printer is a device from the user's perspective. However, it also is a container since it can queue several files. A shredder is a device which deletes the objects dropped on it. A diskette is a container which stores the objects dropped on it.

The objects found in the Workplace Shell are discussed in this section.

4.3.1 Device Objects

Device objects have certain common behaviors; they perform some physical action on an object dropped on them. Objects such as containers and data also perform an action on the objects dropped on them, but the user does not normally perceive a physical connection between the icon and a "real-world" object.

4.3.2 Container Objects

As the name implies these objects are organizational helpers. As in an office, a container holds the objects you are dealing with. They can store any WPS object, including other containers. The standard container in the Workplace Shell is the folder. A folder icon looks, by default, like a manila (cardboard) folder. They are often "personalized" by the user or administrator for easier identification on a cluttered desktop.

Container objects may be folders or work areas. Both types of containers appear and behave in a similar manner; the user can copy or move an object between any container, either folder or work area.

4.3.2.1 Folders

A folder can store and display any kind of Workplace Shell object, but it has a "passive" nature. Even while objects can be opened or started from within a folder, it just stores them and doesn't know anything about them.

4.3.2.2 Work Areas

In contrast to folders, a work area is not completely passive. If an object in a work area has been opened with a different view and the work area is being closed, all the "dependent" views will be closed too.

4.3.3 Data Objects

Data objects may be text files, drawings and spreadsheets. These are said to be of different "types." The object type can be set by the user and is used in a variety of operations such as sorting and assigning associations. Data objects are associated with an "object handler"; this is a generic term for a program that is used with the object.

When the user double clicks the mouse on the icon, the contents of the object are displayed by the object handler in a window. When the window is closed, the object handler saves its data and the location and state of the window for the next time it is opened.

When an object is dragged to a container it is moved there. When it is dragged to a device, the default action depends on the device; the contents are copied to a printer or diskette, but moved to a shredder. When an object is dragged to another data object, the result depends upon whether the target object knows what to do with the object being dragged. To achieve this, there must be a communications protocol by which the objects may communicate with one another, and an agreement on which commands are understood. These considerations are discussed in Chapter 7, "Presentation Manager and Workplace Shell Application Development."

4.3.4 Reference Books

The *Information* folder contains the reference books that were selected during the installation of OS/2 V2.0 and any other materials added later. The installation process allows the user to select from:

- Command Reference
- REXX Reference
- Glossary.

4.3.5 Program References and Shadows

Many of the items a user works with are program references and shadow copies of objects. This arrangement allows him to have objects in many places and still use them as if the original was used. The Workplace Shell supports this concept through various functions like drag and drop, menu selections and object settings. The user has to make sure that this support system is not violated through copies and deletions from the command line or from a program.

4.3.6 Drives

The *Drives* icon opens to a view of all disk drives in the system; these are called *Drive* icons. After "login" to the LAN, any network drives assigned to the user will also be shown here.

Drive icons are container objects which display the contents of the directories available to the user. A *Drive* object represents the "disk partition" and is, therefore, an abstract, non-copyable object. When a *Drive* icon is opened, the same *Drive* icon is seen within it; the difference is that the icon now represents the root directory of that partition and is copyable. Thus a root folder can be copied but a disk partition can't.

The following figure shows the drives icon and two drives folders, one before and one after Login.

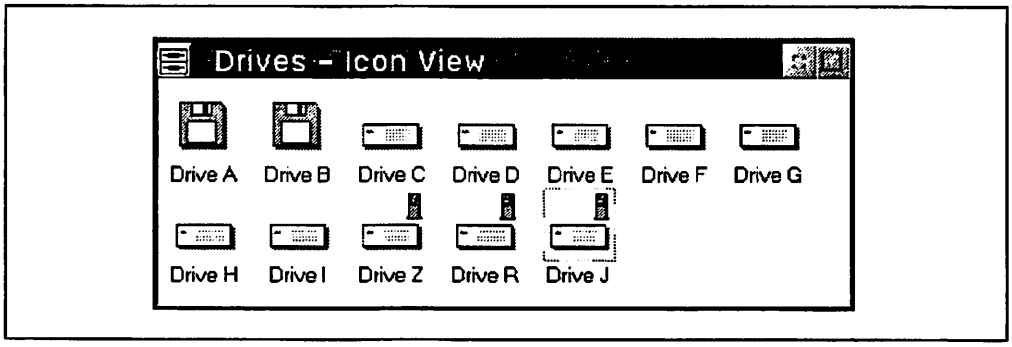

Figure 18. Local Drives and LAN Drives

Directories contain, as before, data, programs and so on. Information pertaining to some of the files may, however, be stored in the OS2.INI file as described in Chapter 8, "Workplace Shell Implementation" on page 111. Copying or deleting a file is therefore a non-trivial exercise for OS/2 V2.0 and should be done through the Workplace Shell to ensure integrity. The folder mechanism allows files to be moved and copied within the subdirectories of the main desktop directory. The *Drive* icons provide the equivalent mechanism for all the directories accessible to the user.

More information can be seen through the *Settings* view. This makes it easy for the end user but harder for the administrator who now needs to understand much more about how the system works.

The *Drives* folder is, partially, what the File Manager was in OS/2 Version 1.3. The main differences stem from the object-oriented view which is different from the hierarchical nature of the PM File Manager. The only hierarchy visible is in the tree view of a drive or a folder.

"Sort" can be selected in both the details and icon views. The sort criteria here are different from the purely filename-oriented view of the File Manager because the extension of a filename has limited importance. The *type* of a file is determined by either OS/2 V2.0 or by the user. It can then be used to sort the folder contents.

For example, when "sort by type" is selected, all executable files are in one group and are sorted by name within that group. This block can thus include files with the extensions EXE, CMD and COM mixed within it.

The drives folders do not display only the files contained in a directory. They also show any shadow copies or program references which may exist in the folder associated with that directory. Folders are linked to subdirectories of the main desktop directory. This information is stored in the OS2.INI file, as described in 8.5, "The OS2.INI File" on page 126, and in the directory Extended Attributes (EAs). This applies to both local objects on the workstation and objects on the LAN drives used by the workstation.

4.3.7 The Shredder Object

The Workplace Shell includes a shredder object, which allows a user to easily delete an object from the system. The shredder object is represented on the desktop by an icon resembling a paper shredder, as shown in Figure 19 below. An object can be deleted by dragging the object over the shredder icon and dropping it there.

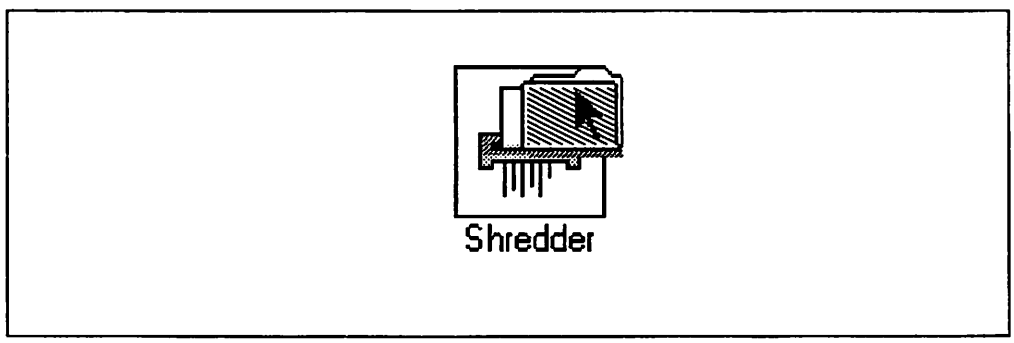

Figure 19. Shredder Object With A Folder For Deletion

Objects may, of course, also be discarded by selecting the *Delete* option from any *File* or *Object Class* pull-down menu.

4.3.8 Printer Objects

Within the Workplace Shell, each possible print object is represented as a separate printer icon. The print object is composed of the printer queue and the ports serviced by that queue. However, the joint nature of the printer object is not apparent to the end user, who sees *only* the printer object, both when installing and when using a printer. This removes a major source of confusion with previous versions of OS/2.

Each printer object has an object name and is represented on the desktop by an icon. A user may direct objects to be printed by dragging and dropping the required object over the printer object.

A printer object may be opened, and displays a window containing the name and status of each print job currently in the queue, as shown in Figure 20. The user may select any job and view information on that job, hold or release the job in the queue, or discard the job from the queue. A printer object may be accessed for drag and drop operations when opened, as well as in its closed icon state.

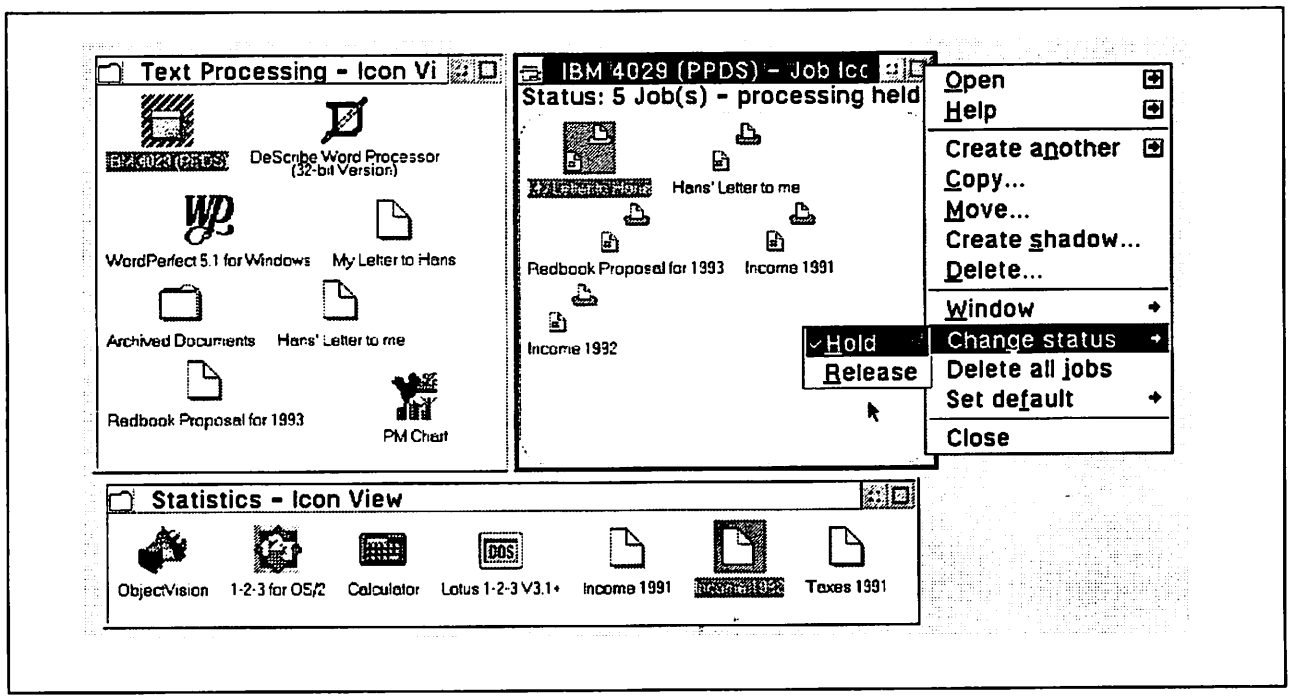

Figure 20. Job List View of Print Object

The printer object window includes a menu with items allowing the user to view and alter properties of the printer object. Selecting an option from this menu causes a notebook control to be displayed, allowing the following items to be viewed or altered:

- Printer port used by the print object
- Name of the printer driver
- Printer options
- Queue options
- Network options
- Pooling options.

In order to create a new printer object, a user may select the *Copy* option which creates a new printer object identical to the current printer object. The desired properties may then be altered and the new object renamed to reflect the changes. For example, the original may print "portrait," but the copy can be changed to print "landscape" on the same printer.

For more information on printing, refer to *OS/2 Version 2.0 - Volume 5: Print Subsystem.*

4.3.9 Templates

A very important part of the object-oriented philosophy is that the user does not load a program and subsequently create a file from the program. Instead, the user works with objects he creates by "cloning" from an existing template. The Workplace Shell allows the user to create unique templates for the different file layouts and use them with the same application. This is discussed further in 6.9.3, "Setting up the Users Work Area" on page 87.

4.3.9.1 Contents after Installation of OS/2

The templates folder contains several common templates like data files, bit maps, folders, and icons which can serve as models for the user's objects. A template behaves like a pad of paper that you can drag a page off but the pad never ends.

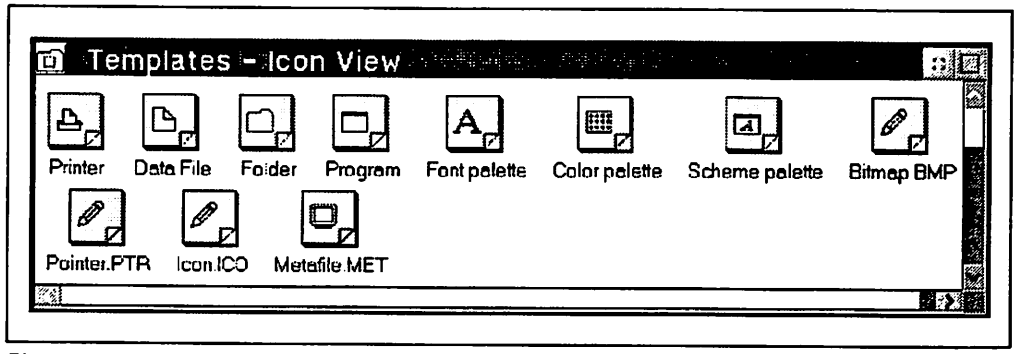

Figure 21. Templates After Full Installation

4.3.9.2 Changes a User Can Make

Users can create an object from the template that comes closest to the type of object that is needed. This object then can be modified to conform precisely to the users' needs and made into a new template.

For example, a "data file" can be dragged from the data file template and put into a work area. The file is then modified to the company memo layout and the standard headings and text are entered. The modified object is then made into a template from which new company memos can be created.

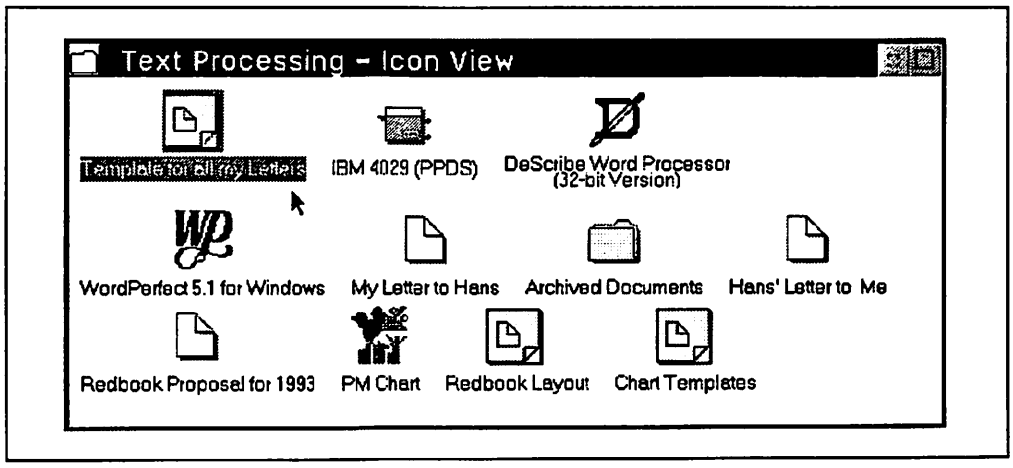

Figure 22. Example of a Folder With User Templates

The administrator has to make sure that the program that is intended to work with this template has the appropriate association. Now, when an object has been created from the template and has been named properly, a double click on the object starts the program to work with that object. Figure 23 shows an object derived from a user template with a cascaded menu showing the association with a program.

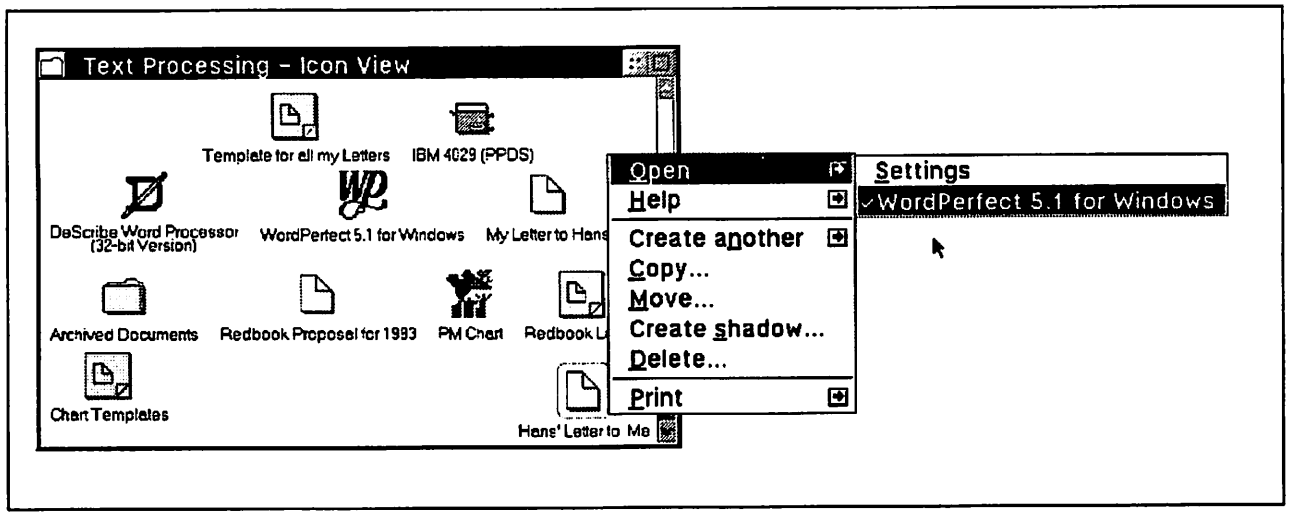

Figure 23. Example of a Menu Showing the Association

4.4 The LAN Independent Shell

The LAN Independent shell is part of OS/2 V2.0 and an integral part of the WPS. If you have one or more requesters started in your CONFIG.SYS, an icon appears on the desktop labelled *Network*. Opening the *Network* folder shows an icon for each network type, such as *LAN Server* or Novell Netware**.

The objective of the LAN independent shell is make the use of LAN resources as simple as possible while still giving users the information they require. There is no longer any need for a user to switch to the full screen interface of the LAN Requester to use LAN resources, they are now integrated into the Workplace Shell

Access to the LAN is provided by a series of folders:

- *Network* folder on the desktop
- *LAN Server* folder(s) controlling the access
- *LAN Server(s)* structure within the network.

The LAN independent shell is loaded into the Workplace Shell when the LAN Requester is installed. The system indicates to the user that the LAN resources are available by displaying the *Network* icon. Any folders that the user does not have access to are not displayed.

The *LAN Server* folder settings show whether a server can be accessed with or without "login." Figure 24 shows the relationship of the *Network* and *LAN Server* folders to the LAN servers and the available folders within them. The LAN folder structure is similar to the structure on the user's own local disk.

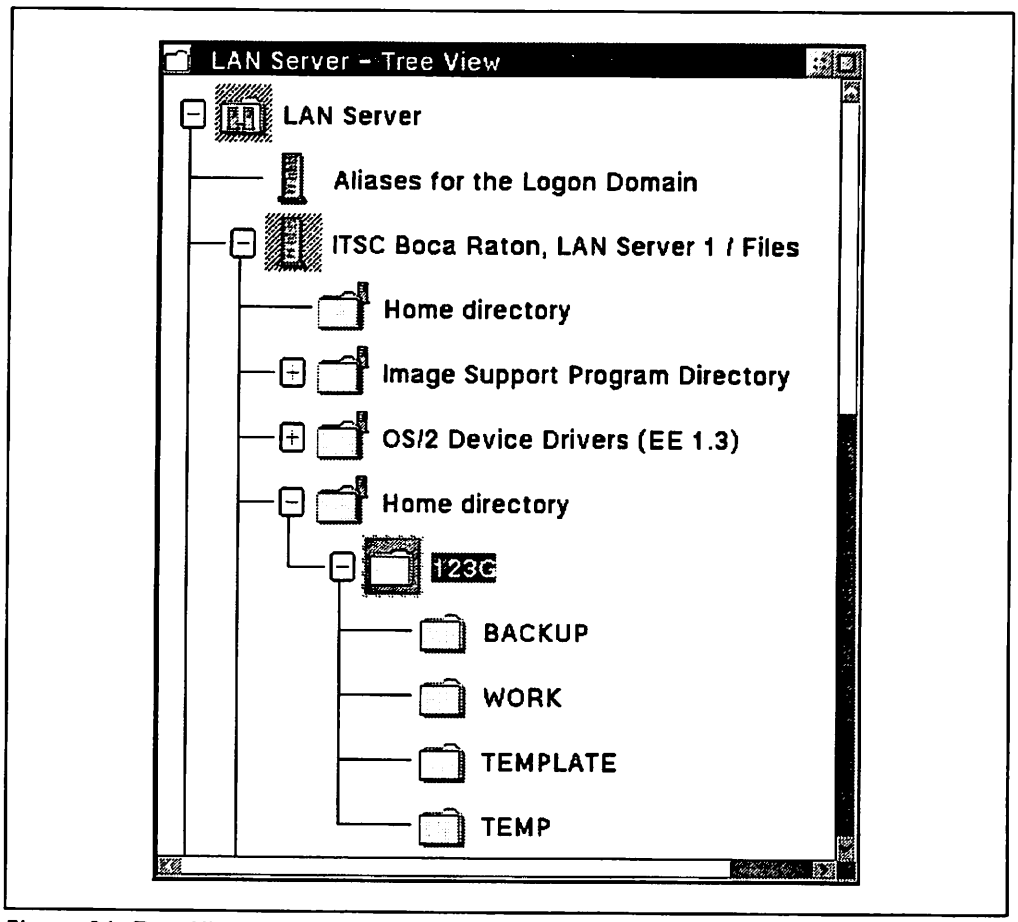

Figure 24. Tree View of the LAN Server

4.4.1 Using the LAN Independent Shell

Opening the *LAN Server* domain icon shows all the servers you are connected to. However, you cannot open this folder until you are logged on. If you try to open it before being logged on, you are prompted for a "user ID" and password before this folder will be opened. Each server can then be opened to show the network resources of either disks or printers. The *Public Applications* folder is provided by the LAN Server and not the shell, hence it is just placed on the desktop.

LAN disks behave just like local drives and the LAN printers behave like local printers. You can drag any network objects onto your desktop to be used later or the next time you start your machine; this saves going back to the network folder. Note that this process creates shadow copies of these objects, not real copies; these will be grayed out until the "login" is completed.

Now you have folders, files and printers that behave just like local objects. All the standard shell operations of drag/drop, move, copy, shadow, opening and settings are available. For example, you can associate a program on the network with a data file on your local disk or network disk. The only restriction is that if you do operations on a network disk that affect the contents of that disk, then you need Read/Write (R/W) permission for that disk.

There is one aspect of the LAN independent shell which may appear as an inconsistency to some users. While you can see any program and data files on remote servers, you cannot see program references or shadow copies within the server. This is due to the design of the Workplace Shell. The WPS only knows about shadows and program reference objects stored in its own OS2.INI file. The WPS on your system cannot read the OS2.INI file on another (remote) system. Therefore any shadows or program references defined within that remote system are rendered invisible to it.

For example, if a user could access the root drive of an OS/2 V2.0 LAN Server, he could use *Drives* to view the desktop structure on that system. However, the *System Setup* folder would appear to be empty. This is because the objects stored in that folder are program references, that is, they are *pointers* to programs in the OS2.INI file on the server.

See Chapter 8, "Workplace Shell Implementation" on page 111 for a detailed explanation of how Workplace Shell objects are implemented.

4.4.2 Main Functions

The LAN independent shell allows you to move or shadow the *Network* folder in any other folder. It has the following functions and features:

- Ability to access multiple networks at the same time
  - IBM LAN Server requester for 2.0 (requires a LAN Server Version 1.3 or 2.0)
  - Novell Netware requester for 2.0 (requires a Novell Netware Version 2.2, 3.10 or 3.11 server).
- Ability to login/logout to networks/servers or resources through the WPS
  - The appropriate login dialog is displayed (if necessary) before a network object can be accessed
  - This is also available from the login/logout context menu items.
- Ability to browse available servers and resources on the LAN
  - Resources are either shared disks or shared printers
  - To open a resource, double click on the network or server icons required.
- Resources can be moved onto the desktop (or any folder) for easy, convenient use both now and later. This information is not affected by the system initial program load (IPL).

  - Servers and shared disks can be shadowed into any folder including the desktop
  - Shared printers can be moved, copied or shadowed into any folder including the desktop.

- Seamless access to network folders and files
  - Disk resources can be opened to show folders and files (which behave just like those in the regular shell). The user can use programs or data files from this network disk.
  - Some applications may not be able to accept the universal naming convention (UNC) filenames provided by the server, such as \\SERVER\DISK\MYFILE.DAT.
- Seamless access to network printers
  - Printer resources can be opened to show queued jobs and job status
  - Printer configuration automatically is set up on the requester's system (the user only needs to install a printer driver as required)
  - The user can only manipulate (hold, release or delete) his own jobs
  - The administrator can manipulate all jobs and hold or release the printer.
- Network printers can be defined to be the default printer
  - No need for users to have local printers defined. If you use the COPY command to print to a Novell server by assigning a port, you must ensure you use the port name \DEV\LPT rather than just LPT*n*, where *n* takes a value between 4 and 9.
- A drive letter or port name can be assigned to a network disk or printer.
  - Disks with assigned letters (for example X:) appear in the *Drives* folder
  - Most applications are **not** network aware and cannot be run from a UNC path (use the "assign drive" context menu item instead).

4.4.3 LAN Server and Novell Netware Support

In order to run both LAN Server and Netware requesters in the same machine the following configuration file changes are required:

C:\CONFIG.SYS ensure DEVICE and RUN statements are in correct order

C:\NETWARE\NET.CFG use the correct protocol bindings for Novell

C:\IBMCOM\PROTOCOL.INI use the correct protocol bindings for LAN Server.

The COEXIST.TXT file that is shipped with the Novell requester provides more information on this.

4.4.3.1 APIs for the LAN Independent Shell

The LAN Independent API is currently supported by Novell Netware and IBM LAN Server.

4.5 Summary

This chapter discussed many characteristics of the Workplace Shell, including the main WPS objects, the LAN independent shell objects and how they are combined to provide a comprehensive and consistent user interface.

There are three main classes of objects within the Workplace Shell: devices, containers and data objects. Within each class there are many types or "subclasses" which have specific features to enable them to perform specialized functions. For example, both folders and work areas are a type of container but work areas have a unique characteristic of being able to close any programs which were started from them when the work area is closed.

LAN integration in OS/2 V2.0 is now much easier, thanks to the LAN independent shell. Access to LAN resources is seamless, and objects can be accessed on the LAN, and then acted on, as if they were local objects. Several new icons, such as the *Network* folder, are placed on the desktop when LAN support is installed.

Chapter 5. Using the Workplace Shell

To use the Workplace Shell (WPS) a user must understand the topics covered below. The user must know how to select an object, how to open a view (or window) of that object, and how to perform some basic actions on that object, such as printing, copying, creating, tailoring, and deleting.

Most importantly, however, the user must start thinking about *objects* instead of *applications.* The metaphor behind the Workplace Shell is data-centric rather than application-centric. Once the object/data is found, the operating system does the rest. Some of the navigation and operation can be done using the keyboard but it is almost impossible to work without a mouse.

5.1 WPS Navigation and Techniques

This section discusses the main techniques used to navigate around the Workplace Shell. The WPS supports user interaction through both mouse and keyboard although it should be stressed that the WPS focus on direct manipulation makes it much easier to use OS/2 V2.0 with a mouse than from the keyboard.

5.1.1 Mouse

Many actions can be performed upon an object simply by using the mouse. Table 1 provides a list of the main shell functions which can be activated using a mouse.

Function	Mouse Button - Action
Select	mb1 - Click
Open icon to window	mb1 - Double-click
Context (pop-up) menu	mb2 - Click
Edit icon description	Alt - Down *plus* select
Drag	mb2 - Down on source
Drop	mb2 - Up on target
Drag and drop	mb2 - Down *then* mb2 - Up
Create on drag	Ctrl -Down *plus* drag and drop
Create shadow	Ctrl - Down *plus* Shift - Down *plus* drag and drop
Change desktop scheme	Alt - Down *plus* drag and drop
Window List	mb1 *plus* mb2 - Click simultaneously

Table 1. Mouse Button Settings. mb1 = mouse button 1, mb2 = mouse button 2

Note: To click simultaneously on mb1 plus mb2 is called "chording"

5.1.2 Keyboard

Though the graphical interface encourages the use of a mouse, the keyboard is still the main data entry device. Once much of the work is done using the keyboard in applications like word processing the user may find the keyboard much faster and easier to use than the mouse.

Besides the convenience aspect, some users may have a special need to operate OS/2 from the keyboard. This may be helpful for handicapped people or in a certain environment where two or more keys cannot be depressed at the same time. This is supported by the so-called *sticky keys.* The sticky keys allow the use of several keys in a sequence instead of pressing them at the same time.

Take, for example, a user who needs to press Alt+F5. With the sticky keys that can be done using one hand and pressing the keys sequentially within a time frame which can be set in the System Setup function.

5.1.3 Accelerators

CUA recommends that applications are written with both mouse and keyboard use in mind. To facilitate this, menus should be accessible through **accelerators.** Accelerators allow the user to access menus by depressing combinations of keys, usually Alt plus some letter. They also permit selection of a choice from a menu by a single letter.

This does not necessarily work in applications that use those key combinations themselves. If, for instance, a menu provides "File" in a menu, then the capital F cannot be used within the application for a function like "Find." In those cases either the key combinations in the application have to be modified (often possible in a profile) or the mouse or function keys have to be used. Please note that uppercase and lowercase characters may be treated differently. This is one of the reasons why CUA demands that applications respect a number of definitions, for example F1=HELP.

5.1.4 Object Manipulation Techniques

Any movement of the mouse or any key causes some message that usually results in some selection or manipulation. Manipulation can be a copy, a change in appearance, an activation, or some other form of action.

5.1.4.1 Traditional Approach

Though the move from the command-line interface to the full-screen interface (with or without drop-down menus) changed the way of operation to an object-action sequence of work, the Workplace Shell is even more different.

Let's take two examples.

1. Copy a file. This involves:
   - Marking a file
   - Activating a menu or function key
   - Naming the destination
   - Performing the operation
   - Updating references, profiles, paths, and other related data.
2. Change colors. This requires the user to:
   - Select a function from a menu
   - Define the appearance of the screen or window
   - Activate the changes for the whole application.

5.1.4.2 Drag and Drop

The most obvious way to act on an object is by dragging and dropping its icon using mouse button 2, sometimes called "direct manipulation." The user presses mouse button 1 to select objects and mouse button 2 to drag and drop them. The user simply moves the cursor to the object to be manipulated, depresses mouse button 2 to start the drag, holds the mouse button down while moving the cursor to the target, then releases mouse button 2 to drop the object.

Using drag and drop to perform the same actions described above takes just a few simple steps in OS/2 V2.0:

1. Copy a file:
   - Make the file to be copied visible
   - Make the destination visible
   - Point with the mouse at the file object, depress mouse button 2 and drag the object to the destination
   - Drop the object.

   OS/2 does almost all changes to related data for you, such as changing information about a shadow copy. Some information, like a changed path, has to be updated manually.
2. Change colors
   - Display the color palette or the scheme palette
   - Display the object to be colored
   - Drag the color icon or the scheme icon to the object and drop it.

   The coloring can be done on an object basis or on an application basis.

If an icon is dragged to a printer, the contents of the object are printed. If the object is a file, then the file's contents are printed; if it is a folder, then a list of the folder's contents is produced. In the same way, dropping an icon on the shredder deletes the object, while dropping it on a folder, workplace or diskette moves it into that object.

The concept of direct manipulation implies that certain types of objects may be dropped in certain locations. For example, a printer object cannot be dropped over another printer object, since it is logically impossible to print a printer. It is therefore necessary to provide some form of indication as to whether a "drop" is allowed on a particular object.

This is done visually, by altering the appearance of the icon representing the object being dragged, whenever a drop operation is *not* permitted. When the icon is moved over an object or location where a drop *is* permitted, the destination object shows a box around it. However, this does not mean that the target object will actually be able to correctly handle the object to be dropped.

As some of the functions cannot be done with just the two mouse buttons it may be necessary to use the Alt-, the Ctrl- or the Shift-key as a *modifier.* For example, depressing the Ctrl-key when dragging an object causes a copy instead of a move operation.

This is discussed in more detail in Chapter 7, "Presentation Manager and Workplace Shell Application Development" and Chapter 8, "Workplace Shell Implementation."

5.2 Basic Operation of the Workplace Shell

This section describes how to perform some of the basic tasks available to Workplace Shell users.

5.2.1 Accessing a Context Menu

All objects have a context menu which is activated by pointing at the object and depressing mouse button 2 (this is also known as "popping up" a menu). These menus are contextual because the content depends on the capabilities of the individual object. They can often be extended by the user so new functions can be added to the object.

Usually these context menus reflect the capabilities of a single object. One has to be careful when selecting several objects and then activating the context menu of one of the objects. This menu then shows only those choices which pertain to **all** of the selected objects.

5.2.2 Opening a Window

The user may view the contents of an object by double-clicking on the icon. Two things then happen: the icon background changes to inform the user that it is open, and a window appears with the contents displayed. Most objects also support different views; for example, a folder may have a *Details* view, an *Icon* view and a *Settings* view where the user may customize certain features.

A window in OS/2 V2.0 looks almost like any window in OS/2 Version 1.3, except that there is a different icon (called the small icon, mini-icon or title-bar icon) instead of the system menu icon. This aids visual association of the icon with the object on the part of the end user. This icon does not, however, support the same drag and drop actions that can be performed by using the icon on the desktop. The "old" minimize button has been replaced with a choice of a new minimize button (a small square) or a "hide" button. The maximize button is now a large square.

The user may also open a window by using its **context menu**; this menu is triggered by clicking mouse button 2. The user may then select the *Open* choice from the menu.

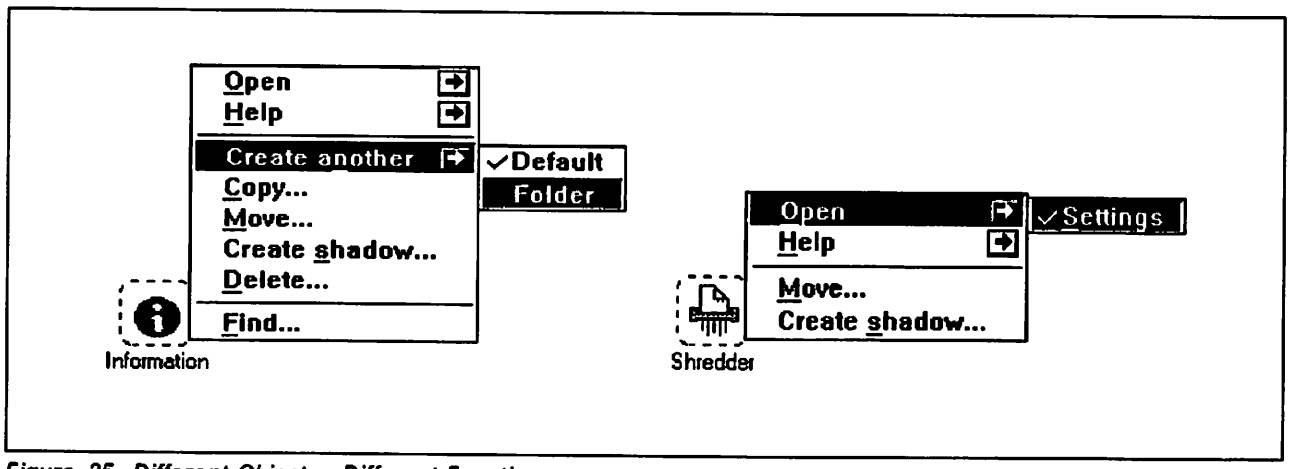

Figure 25. Different Objects - Different Functions

This results in a cascaded menu being displayed. In the case of a folder the user can typically choose to look at the *Settings*, *Details* or *Contents* view of the object. This is an important characteristic of the Workplace Shell; the ability to work with different views of the same object helps the user to look beyond the windows and icons on the desktop to see the underlying objects and their relationships to real-world objects.

The *Details* view looks similar to the OS/2 Version 1.3 File Managers list style. The *Icon* view is used by folders to show the objects inside them. All objects have a *Settings* view which is used to change the objects' properties.

5.2.3 Finding Open Windows

With Presentation Manager under previous versions of OS/2, when a window was minimized it would be shrunk to an icon and placed on the desktop. The user could restore the window either by double-clicking mouse button 1 on its icon, or by using the Task List. When the window was restored, the icon disappeared from the desktop and was replaced by the window.

Under the Workplace Shell, when the user opens an object its icon remains on the desktop, but its background changes to show that it is open. When the user closes a window, the icon background changes to show that there is no longer an open window.

If the window is minimized, the window disappears but the icon background does not change; this shows that the window is still open but has simply been "hidden." The system settings allow the user to choose between "hiding" and "minimizing" a window. There are a further two options for the minimize function.

The first minimize option puts a *Minimized Window* folder on the desktop so that users can access any minimized windows from it, while the second puts the minimized windows directly on the desktop. This latter approach is more compatible with OS/2 Version 1.3 but may prove confusing to users.

The Workplace Shell has replaced the Presentation Manager Task List with a **Window List.** The Window List represents all objects which are currently open or active, and allows the user to find and restore windows which are open but hidden. The Window List can be accessed at any time by chording the mouse buttons over the desktop. Subsequent selection of a single line with mouse button 1 and opening of the context menu with mouse button 2 offers various choices to select the desired view of the object.

5.2.4 Creating A New Object

To create a new object, the user can either use a template or simply make a copy of an existing object and change it. Templates may be regarded as pads of blank forms; the user simply "peels off" a new instance of an object such as a folder. The Workplace Shell creates a new icon of the requested type, which the user can then customize using the *Settings* view.

To make a copy of an existing object, the user may either bring up a context menu and select the *Copy* choice from it, or use the *Create-on-drag* function by holding down the control key while dragging the icon to the required destination.

5.2.5 Creating a Shadow Object

An interesting new feature is the ability to make **shadow copies** of existing objects which can be placed in different folders or locations. These shadow copies are linked to the original so that no matter which copy is changed, all copies are automatically updated. Links between shadow objects are automatically maintained by the Workplace Shell. These links are sometimes referred to as **reference links.**

The user may create a shadow copy from the icon's context menu or by holding down the Ctrl and Shift keys while dragging the icon to the required destination. If a shadow is created in this way, the system draws a line between the original and the shadow until the icon is released, so that the user knows that a shadow copy, rather than a "normal" copy, is being created.

Program files should not be "shadowed" to a folder on the desktop. Instead, a new reference for that program should be created within that folder. This is done by opening the *Templates* folder and dragging a program object to the folder you wish to run the application from. This is described in the online reference documentation.

Using a program reference to the executable (.EXE) file means that, when the user direct edits (Alt-mouse button 1) the icon description, he will not change the name of the executable file. Changing the filename could significantly confuse the Workplace Shell. This is also described in 6.3.2, "Shadow Copies of Programs" on page 73.

5.2.6 Deleting and Undeleting an Object

Almost any object can be deleted by either dropping it on the shredder or by selecting *Delete...* from the object's context menu. As in previous versions of OS/2, files and directories can be deleted from a command line.

To help the user recover from accidentally deleting files from a command prompt, OS/2 can set up a special *Delete* directory on each logical drive. Each deleted file will be put into this directory until a user-specified size is reached. Only then will the files really be deleted from the disk. This will be done in a FIFO (First In, First Out) sequence.

Files can be recovered easily if they are still in the DELETE directory by just copying them back to where they were. If a file is already erased from the DELETE directory it may still be recoverable with the UNDELETE command.

5.2.7 Printing

The user wants to print specific types of output in an appropriate manner on the right paper. So it should not matter to him how the system gets this done. All that is important is that the output is directed to some object that subsequently produces the desired printout. Therefore the concept of logical printers in a print manager was changed to the concept of printer objects.

A printer object represents a certain arrangement of hardware, software, and specifications that are made to simplify the printing for the user. So, if printing is needed on three different sizes or types of paper then there will be three different printer objects, each for one specific type of output.

5.2.7.1 Local OS/2 Printing

For local printing everything is installed on the individual workstation and therefore it is simple to install a printer and the driver for it. Other printer objects directing different output to the same printer can be quickly created using the Workplace Shell.

5.2.7.2 Printing from DOS Programs

The term *DOS Printing* refers to two different types of printing:

1. Printing from an OS/2 session with DOS-like commands (that is, PRINT, TYPE, PrtScrn)
2. Printing from DOS applications using DOS device drivers.

As DOS printing does not happen from the Workplace Shell there are also no icons to represent any destinations. This also means that these functions are done in the traditional way by setting up a printer from within applications.

DOS Programs are discussed in detail in *OS/2 Version 2.0 - Volume 2: DOS and Windows Environment*.

5.2.7.3 Remote Printing

The process of remote printing can become a little bit more complicated because the setup for the destination printer is done on the print server. For the local user, printer objects have to be created that reflect the functions on the server. In addition, the local printer object has to indicate the status of the remote printer.

Printing is described in more detail in *OS/2 Version 2.0 - Volume 5: Print Subsystem*.

5.2.8 Creating a Startup Environment

In many cases a special setup for users is required. This setup can be created easily by putting shadows of programs to be run (for instance Communications Manager) into the *Startup* folder. This folder is opened when the operating system is initially loaded (known as "initial program load" - IPL - or "boot") and the applications in it are then started.

Though a shutdown remembers the running applications and restarts them after booting the system there is still a need for the *Startup* folder. Shutdown only remembers what was running at the time, whereas the *Startup* folder has a fixed set of applications to start no matter if they were running at shutdown or not.

5.2.8.1 Startup Folder Sequence

It is also possible to order items in the *Startup* folder. First, display the folder using an ordered view (that is, anything except icon view non-grid). Then drag items into the folder in the required order. This can be useful for programs which are dependent on others being started before they are started.

5.3 Advanced Operation of the Workplace Shell

This section describes how to perform more advanced tasks for Workplace Shell users.

5.3.1 Changing the Characteristics of an Object

The *Settings* view for an object can be used to change characteristics such as color, font and icon bitmap, as well as other application-specific properties such as use of a menu bar or a context menu.

The user may also change some attributes such as color and font by opening the color and font dialogs in the *OS/2 System* folder and dragging the desired color or font to the object. If the color or font is dragged to the *Templates* folder, then the default settings are altered for all new objects created from the templates.

As well as allowing the user to change the attributes of an object to be different from its parent, it also allows him to undo this so that the object inherits from its parent again. For example, if a color scheme is dropped on a folder, that folder will differ from the desktop. If a different scheme is then dropped onto the desktop via Alt-drag, that folder's attributes do not change.

To get the folder to forget its color settings and inherit the desktop color scheme again, just drag the same scheme that was used for the desktop onto the folder with Alt-drag, that is, the same mechanism that was used to set the desktop colors. The folder will release its presentation parameters and accept the system default parameters which have been dragged to it. Any future changes to the desktop color settings will now be inherited by that folder.

5.3.2 Modifying the Context Menu of an Object

The experienced user may be able to add items to the context menus of selected objects. The procedure is basically the same for any kind of object. It is described here using the desktop menu. This is how the menu looks after the modification:

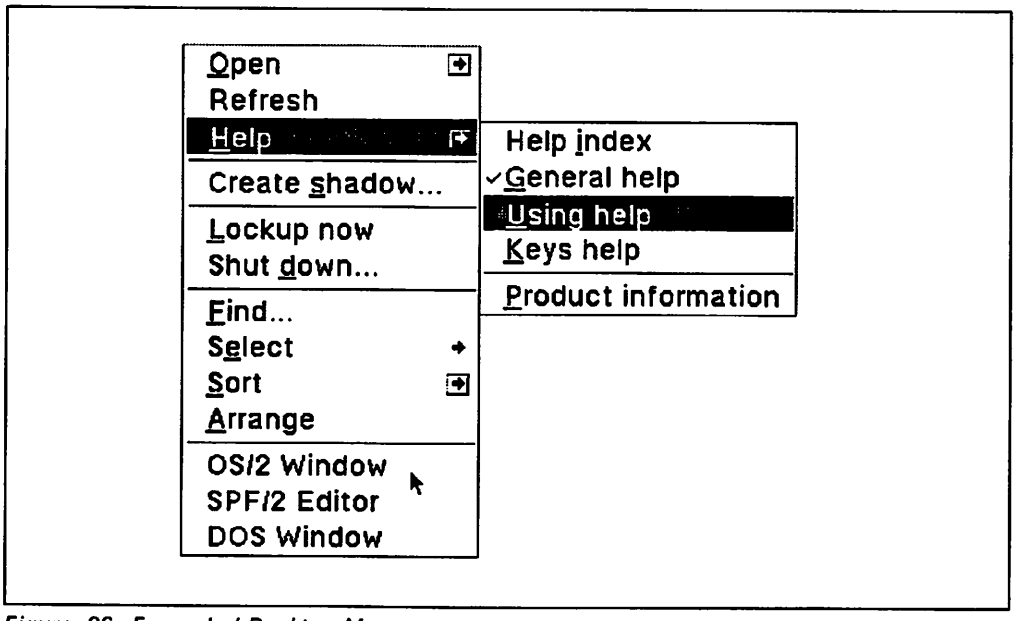

Figure 26. Expanded Desktop Menu

All the modifications are made through the *Settings* view of the object. For the desktop, this is accessed by clicking mouse button 2 on the desktop background.

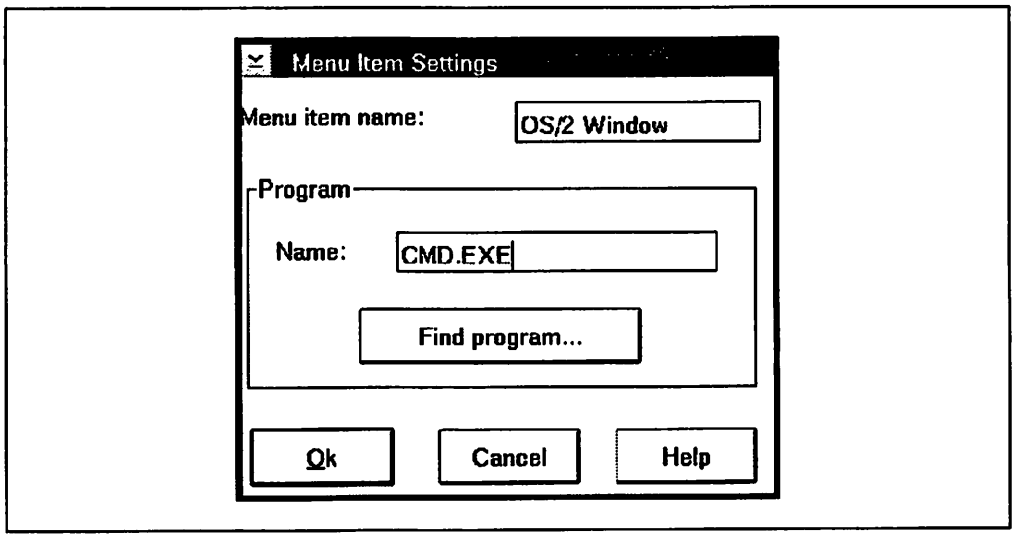

Figure 27. Setup of an Expanded Menu

The following sequence will add "OS/2 Window" to the context menu of the desktop. When this choice is selected, an OS/2 Window Session will be started.

Step 1. In the upper part of the notebook page ("Available menus") select "Primary pop-up menu."

Step 2. In the lower part of the page ("Actions on menu"), select the "Create another..." option. This will bring up the window shown in Figure 27.

Step 3. Enter "OS/2 Window" in the "Menu item name" entry field.

Step 4. Enter *CMD.EXE* in the "Program name" entry field.

Step 5. Close the "Settings" menu.

Step 6. Activate the menu and the newly added entry to verify that it was successful.

If the "Create another" choice under "Available menus" is selected then a further choice of "Cascade" or "Conditional cascade" is used to determine whether a secondary menu will be displayed.

1. *Cascade* means that the new entry will have a cascaded menu. This is indicated by an arrow.
2. *Conditional cascade* means that there will be a cascaded menu with a default selection. This is indicated by a button with an arrow.

5.3.3 Changing the Default View on "Open"

The Workplace Shell allows users to change the default view of an object (that is, the view that opens when it is double clicked on). The "Open" entry in a primary menu is usually the only one that allows additions because it is application-oriented. Once "Open" or one of the added entries is selected, the cascaded menu of this entry is addressed by the lower part of the notebook page. Here the actual program names have to be entered under "Actions on menu." After entries have been made here it is recommended that an existing choice be selected as the default option.

For example, a user might want to change the *Drive* folder and objects so that the default view is "Details" instead of "Icons." The following steps are used:

Step 1. Open the *Settings* view and go to the "Menu" page. This page contains "Primary pop-up menu" and "Open" headings.

Step 2. Select "Open," then press the adjacent "Settings" button.

Step 3. Select the cascade arrow next to "Default action," then pick the new default view you want to use when you subsequently double click on the icon.

Step 4. Click on the "OK" push button.

Step 5. Close the *Settings* view.

5.3.4 Changing the Icon for an Object

The Workplace Shell makes it easy to customize the appearance of the objects the user needs to use. The icon description can be changed by selecting it with "Alt-mouse button 1" or the icon and its text can both be changed from its *Settings* view.

The "General" page of the objects *Settings* view gives you two ways to change the icon: *Find* and *Edit*. Find lets the user choose a new icon from those already created. Edit starts the icon editor to let you work with the icon; this can be used to modify the existing icon or to work with a different one.

Please note that you must take care to edit the icon belonging to your particular "device"; that is, 32x32 and 16x16 icons for VGA/EGA, 40x40 and 20x20 for higher resolutions. The "Device" pull-down in the menu bar of the icon editor will let you do this.

The Icon Editor is very flexible. A user can:

- Paste in a new icon from the clipboard, providing it was placed on the clipboard before the Edit action was started
- Use the icon editor menu bar to:
  1. Select "Open..." to choose a new icon
  2. Select "Save as..." to save it under the *original* filename for the old icon that was first loaded into the icon editor (this is usually x:\OS2\WP!*n*.ICO, where *n* is a number).

When you have finished, close the window and respond "yes" to the prompt to save the changes.

5.3.5 Associating an Object with a Program

There are three ways to set up a data file object so that a program is started when the user double clicks on the data file icon. These are:

1. Create a *Filename* association in the program settings
2. Create a *Type* association in the data file settings
3. Make the program name the default setting in the "Open" cascade menu for the data file.

An association is a special link which can be assigned to a program object. This link allows the user to specify which program will be invoked when the user

double clicks on specified types of files, individual files, or files described by a global file name.

5.3.5.1 Filename Association in a Program

The major disadvantages of this approach are that it provides little flexibility to the user and it is not very robust. Any association is easily destroyed by changing the file extension and it is therefore only recommended for programs which do not support associations by "Type."

The inflexibility of this method can be seen when a user subsequently wants to use a different program with his data file. To do this, he will have to either change the filename or add a new program to the "Open" menu and make it the default. Neither approach is suitable for the inexperienced user.

A text editor may, for example, have associations for plain text files (that is, a type of file), the individual file CONFIG.SYS, and all files described with the global name READ*.*. Whenever a file object that meets these criteria is selected with a double click, OS/2 will start the appropriate program object. The workings of the association mechanism are described in Chapter 8, "Workplace Shell Implementation."

There are some editing strings for filenames and extensions that can be used with associations to allow parameters to be passed without having to fully specify the qualified filename (or parts of it):

%*	Fully qualified path
%%*P	Path with last backslash
%%*D	Drive with : or UNC name
%%*N	Name without extension
%%*F	Name with extension
%%*E	Extension without period.

5.3.5.2 Type Association in a Data File

This is the preferred approach. It is described in more detail in Chapter 7, "Presentation Manager and Workplace Shell Application Development" and Chapter 8, "Workplace Shell Implementation." The advantage of this approach is that it is easy for a user to use a different program with his object, simply by choosing a new "Type" in the *Settings* view. The process is simple:

Step 1. Open the *Settings* view for the object

Step 2. Select the "Type" page of the notebook

Step 3. Select from the list of "Available types"

Step 4. Press the "Add" button

Step 5. Close the *Settings* window.

Each program can be associated with a "Type" but, in practice, not all are. DOS programs are very unlikely to support the concept of file type associations. It will, therefore, probably be necessary to use the filename association technique for such files.

If a program does not set a file type, the user can create a new one. This is described in 6.8.5, "Adding New File Types" on page 84.

5.3.5.3 Add a Default Program to the File Menu

The process to do this is described in 5.3.2, "Modifying the Context Menu of an Object" on page 60 and 5.3.3, "Changing the Default View on "Open"" on page 61. This approach is limited to users who understand the workings of the Workplace Shell in some detail. Its main disadvantage is found when many existing files must be set up to use a particular program; each one must be modified separately. Under these circumstances the filename association would be preferred.

If you are setting up a system for a new user, however, you can create one data file with the program set as the default on "Open" in the context menu, then make that file into a template so all new files subsequently created from it will inherit this attribute.

5.4 Giving OS/2 V2.0 the Look and Feel of OS/2 Version 1.3

Users of earlier versions of OS/2 or of Microsoft Windows may not feel entirely at home when they first start the system. The graphical user interface (GUI) they are familiar with has been replaced by an object-oriented user interface (OOUI) which, while providing a clear, logical layout is very different. In general, if a user thinks about objects rather than programs, he will find it much easier to learn to use the WPS interface to its full extent.

One of the big differences is that previously a user could start a new application, browse through the menus and scan the possible choices. This way it was fast and fairly easy to grasp the various functions of a program. With the Workplace Shell, however, many functions are not program-specific functions any more but general capabilities of almost all objects. It also means that certain functions are not needed any more in menus; they just exist as OS/2 functions. For example, the choice of a color or a font does not need to be in the menus of each program, the *System Setup* folder provide all the choices the user needs.

If a user prefers the style of OS/2 Version 1.3 he can customize the system through the use of the .RC files that come with the installation diskettes. The procedure is fairly simple and can be done after the completion of a normal installation:

Step 1. Boot the system from the "Installation" diskette

Step 2. Use diskette 1 as directed

Step 3. Exit from the "Welcome" screen

Step 4. Change drive and current directory to C:\OS2

Step 5. Modify OS2.INI with the command:

```
MAKEINI OS2.INI OS2_13.RC
```

Step 6. Restart the system.

If you want to change back just follow the same procedure but make the OS2.INI modification with the command:

```
MAKEINI OS2.INI OS2_20.RC
```

You should note, however, that the MAKEINI command generates a new OS2.INI file from the resource files which were shipped with the system. This means that the OS2.INI file is restored to the same state as when OS/2 V2.0 was first

installed and any customization performed by the user, either to the Workplace Shell or the PM 1.3 shell, will be lost.

5.5 Summary

This chapter discussed many of the techniques supported by the Workplace Shell.

The primary navigation techniques which let the user work with those objects are slightly different from those used by Presentation Manager V1.3, but are at the same time more consistent. The emphasis on direct manipulation helps provide much of this enhanced consistency but enforces the need for a user to have a mouse if he is to perform many basic tasks.

The new WPS objects and techniques offer greatly increased flexibility for organizing work to suit the user. These objects can be rearranged to provide new ways of doing the kinds of activities most users are familiar with. In particular, direct manipulation allows users to perform activities with greater ease, and better feedback, than before.

The Workplace Shell makes OS/2 V2.0 much easier to "personalize" than previous versions of OS/2. Users can easily change one object, the whole system and even the menus of individual items. The shell can even be made to look like OS/2 Version 1.3 to suit the preferences of those users who are not yet ready to tackle an object-oriented user interface.

Chapter 6. Installing and Supporting the Workplace Shell

This chapter discusses the issues associated with installing a system as complex and flexible as OS/2 V2.0. This is not a trivial topic; we must balance the needs of the user for a productive work environment with those of the administrator to be able to support him if problems arise. This requires careful management.

Among the many things an administrator must consider are:

- Allocating disk space
- Setting up programs and files
- Problem determination and resolution
- Backup procedures
- Use of a LAN from the Workplace Shell
- System performance
- Installing application programs.

This chapter offers some guidance on these and other matters related to the installation, configuration and management of systems which include the Workplace Shell. It also includes an illustration of how the system might be set up for a particular user on his specific hardware configuration.

6.1 Allocating Disk Space

In this section we discuss how the effectiveness of the Workplace Shell may be affected by such considerations as:

- Disk partitions
- Choice of file system (HPFS versus FAT)
- Location of the OS/2 desktop directory
- Program set up
- Placement of data and program files
- DOS programs.

6.1.1 Partitioning the Disk for OS/2 with the Workplace Shell

A commonly asked question is "Is it more convenient to have one large disk partition than several smaller ones?" We do not believe that single disk partitions are the correct approach for OS/2 V2.0 workstations.

Single partitions have certain advantages. They optimize the use of available disk space because both the operating system and applications can use what they need while not leaving unused space in the respective partitions.

They are also simpler to set up. There can be logistical problems with multiple partitions, such as allocating enough space for dynamic system files such as the desktop, the SWAPPER.DAT and print spooler.

On the other hand, there are several disadvantages to the single partition approach.

- Multiple partitions let you keep system and user files separate so that the system partition can be re-formatted if necessary. This can be very important when planning to install a CSD or a new version of the operating system.
- Performance can be impaired when a partition contains lots of directories. For example, opening a tree view can take a long time on a large disk.
- Support for multiple operating systems or versions of the same operating system requires multiple partitions to be manageable.

6.1.2 HPFS or FAT Format?

The Workplace Shell introduces several new reasons for choosing the High Performance File System (HPFS) instead of the File Allocation Table (FAT) format for your most heavily used disks and partitions. The main ones are:

6.1.2.1 Performance

Although the FAT file system is much faster in OS/2 V2.0 than it was in OS/2 Version 1.3, there are still areas where HPFS is faster. One major advantage is the reduction of disk fragmentation, discussed below. Another is the performance when reading large files; the "EA_DATA. SF" file on a FAT file system can exceed 600 KB, which could impair the performance of instantiating WPFileSystem objects.

6.1.2.2 Fragmentation

Since the WPS encourages users to move files around and create new folders (directories), the file system is more heavily used than it would be under DOS. Past experiences with heavily used DOS workstations and LAN Servers leads us to feel there is a distinct possibility that fragmentation will cause performance problems on FAT-based disks.

6.1.2.3 Use of Extended Attributes

Since Extended Attributes (EAs) are used extensively by all directories and files within the desktop structure, there are important considerations for file transfer between the different file systems. These EAs store settings information, such as file type, without which the system cannot function properly. In general, we recommend installing a common file system on all machines to prevent potential problems with lost EAs. EAs are discussed more fully in 6.2.1, "Extended Attributes" on page 70 and 8.4, "Extended Attributes" on page 124.

6.1.2.4 Long Filenames

The WPS allows a user to rename a file by editing the icon description. On HPFS, this name will be stored as it is typed, including spaces. On FAT, however, the name is truncated by removing vowels. Not only might this cause problems with duplicate filenames, it also introduces an inconsistency between what the user sees in a folder and in drives which would not otherwise occur with HPFS.

6.1.2.5 Support for Multiple Operating Systems

Multiple operating systems may be required by some users. This would lead to the user installing different file systems on the various partitions to support the different operating systems. For example, a user might need to be able to boot from DOS occasionally, to run programs that don't work in a VDM. To do this he could format the C: partition as HPFS, for OS/2 V2.0, and the D: partition as FAT, for DOS.

6.1.3 Keeping the Desktop Separate from the System

When OS/2 is installed it sets up a directory called *OS!2 2.0 Desktop* on the boot drive, corresponding to the desktop work area. In this directory it installs some default objects for the user which the user can work with to subsequently create new folders and other objects for himself.

The result is that user files are created on the same partition as the operating system. Many users would prefer to keep their data and programs in a separate partition from the operating systems, in case they have to format their partition to install a new release of the operating system.

Fortunately, it is possible to move the desktop to another drive or partition, though this can only be done after installation - the OS/2 install program always puts it on the boot partition. The process to do this is simply:

Step 1. Open the Drives object

Step 2. Open a view of the drive on which the desktop currently resides

Step 3. Drag the "OS!2 2.0 Desktop" directory object to the drive you want it to be on.

This will result in the desktop structure, with all its subdirectories, being physically moved to the new drive or partition. All references in OS2.INI to objects within the corresponding folders will be updated to reflect the change.

If the user does have to reinstall his operating system, however, this approach may cause other problems in rebuilding the desktop. Among the factors to be considered are:

- The WPS will install the desktop into the operating system by default, so the user will now have two desktops
- Any folders created by the user will not be in the new desktop
- Any programs created by the user will not be in the new OS2.INI file
- Programs may be given new HOBJECTs by the WPS as it installs them, so the HOBJECTs in the users directories will now be different from those in the OS2.INI file.

The implementation of the Workplace Shell is discussed in Chapter 8, "Workplace Shell Implementation"; an understanding of the way in which WPS objects are created and stored will help the user to decide how to structure his system.

6.1.4 Moving the Print Spooler

Moving the spooler can be done at any time. The process is very straightforward:

Step 1. Open the *OS/2 System* folder

Step 2. Find the *System Setup* folder and open it

Step 3. Find the *Spooler* folder and open it

Step 4. Specify the spool path in the dialog and close it

Step 5. Reboot and your spooler will be moved.

Remember to check that OS/2 V2.0 deleted the old SPOOL directory when it created the new one you specified.

6.2 Setting Up Programs and Files

There are several issues associated with setting up programs and files. Some of these issues, such as the physical location of programs and data, are also discussed in 6.5, "Using the Workplace Shell in a LAN Environment" on page 76. This section discusses only those issues which affect all programs and files across local and remote disks, such as the use of Extended Attributes (EAs) and file associations.

6.2.1 Extended Attributes

Extended Attributes (EAs) are used extensively by the Workplace Shell. The settings data for any object are stored in EAs. File settings are stored in file EAs while folder settings and some content attributes are stored in directory EA files.

On an HPFS partition, Extended Attributes are stored in a special, hidden area close to the files themselves. On a FAT file system, Extended Attributes are stored in a hidden file in the root directory of each FAT partition; this file is named "EA_DATA. SF"

Not all file systems support EAs and this can cause problems when transferring files around a LAN or to diskette. For example, using files from a server using Novell Netware before Version 3.11, or using AIX*, would lead to the EAs being lost. This is also a problem for DOS programs, which do not understand EAs and therefore cannot write them when they replace a file.

For a more complete discussion of how EAs are used in the Workplace Shell, refer to 8.4, "Extended Attributes" on page 124.

6.2.2 EAs for Files Used by DOS Programs

While the Workplace Shell uses file EAs extensively to store settings information for File System objects, no DOS programs know how to handle them. This can cause problems when a DOS program does not write the EA back to the disk along with the file. Without its settings information, the file will lose information such as a modified icon or the file type.

The problem occurs because many applications do not write back to the same file they read from. Instead, the applications typically perform the following actions:

- When the user selects "Open," the program:
  1. Opens the original file and reads it into memory
  2. Closes the original file
  3. Opens a new, temporary file
  4. Lets the user edit, working with the copy in memory.
- When the user selects "Save," or "Autosave" is invoked, the program:
  1. Writes from memory to the temporary file.
- When the user selects "Exit," the program:
  1. Closes the temporary file
  2. Erases the original file
  3. Renames the new file to the original filename.

Programs do this to reduce the likelihood of the original file being corrupted if the system crashes. The problem is that if the application does not know about EAs, then the above sequence will lose the EAs. All DOS and Windows applications are affected by this, although the problem is more likely to affect programs, such as word processors, which work with files rather than those which work with records within a file, like data base programs.

To get around this, you could create a command file that would use EAUTIL to split the EAs from the file, invoke the DOS application and then, when the DOS application has finished, use EAUTIL to join the EAs back to the file.

An alternative, though long-term, approach is replace DOS versions of these programs with OS/2 versions, which will not have this problem.

6.2.3 Using Files Outside the WPS Directory Structure

Setting up files to be used by the WPS where those files reside outside the "OS!2 2.0 Desktop" directory structure may cause some problems. This could apply, for example, to DOS programs which only look for files in a specific directory. These files can still be placed on the desktop, however. This is done by dragging a shadow copy of the file from the application directory to the desktop, using the *Drives* folder.

However, the WPS does not receive every message from the file system concerning files which lay outside its workplace directory structure (that is, not under "OS!2 2.0 Desktop"). Notification is received if a new file is created or if an existing file is deleted or renamed, but not if an existing file is changed outside of the workplace. This lack of notification also applies to file EAs created or modified for files in a non-workplace directory.

It is therefore always advisable to place all data files within the workplace directory structure, where possible.

6.2.4 Setting Up Programs in the Workplace Shell

Installing programs in OS/2 V2.0 is no more difficult than under OS/2 Version 1.3. The difference only becomes apparent after the program has been installed and the user or administrator is trying execute it. Programs can continue to be run, as in OS/2 Version 1.3, by double-clicking on their program icon. Taking full advantage of the WPS, however, means setting up folders with data files and linking these files to the programs so that they are automatically started when the data file is double clicked on.

There are three main ways of making these links:

- Associate the program with a file type and set the "Type" in the data files
- Associate the program with some or all of a filename and extension to be used by all the data files
- Add the program name to the data files and make it the default program to be used when the user double clicks on the file.

All three approaches have advantages and disadvantages, which are briefly outlined in Chapter 5, "Using the Workplace Shell."

6.2.4.1 File Type Association

Associating programs with files can be done in the programs *Settings* view, by filenames, or through setting the file type in the data file settings. OS/2 provides a default table of available file types to which programs written for the Workplace Shell can add their entries.

So far very few programs take advantage of this feature, even though they could very effectively be started by association if they did (that is, they are written to expect a filename as their first command line parameter).

Where such a program is written by the user, it is a very simple matter to add the required ASSOCTABLE to add the new type. See 7.3.3.3, "Using an ASSOCTABLE to Add New File Types" on page 107 for details of how to do this. In the case of any other program, however, you may want to add a file type to the system, without having access to the source code of the program concerned. A REXX program to do this can be found in 6.8.5, "Adding New File Types" on page 84.

Some users have noted the format in which the table of available types is stored in OS2.INI and have written REXX programs to add new types directly. This approach is **not** recommended; it is unsupported and is dependent on the way OS/2 stores this data, which may change in some future release.

6.2.4.2 Filename Association

The answer for programs over which you do not have control is to use association by file extension rather than file type. This is simple and easily understood by administrators but has some serious flaws and is not recommended as the default solution.

Consider the PMSPREAD spreadsheet program that is provided as one of the productivity "applets" with OS/2 Version 2.0. If you invoke this program with the name of one its saved spreadsheets as the first parameter, it will automatically load the data as the program starts.

If you open a *Settings* view of the program object in the *Productivity* folder, you will see that no type associations are defined for it. So, if we want to start this program by opening one of its data files, then we will have either to add a new file type to the system and associate the program with that, or else associate the program with a filename and/or extension. This particular program uses the distinctive extension *.$$S* for its files, so association by file extension is probably the best approach in this case.

The major restriction with using filename association is, as noted above, that HPFS filenames can be changed by editing the icon descriptions. Unless users remember to add the correct file extension when they edit a filename (and this is not a natural way of working with the WPS) then the program associations will be lost. Some measure of protection is afforded by the "Confirm on rename of files with extensions" option in the *System Settings* folder (it is "on" by default) but this may not always prevent the user from renaming the file.

6.2.4.3 Adding a Program to a Data File Menu

An alternative approach to linking data files to programs is to add the program name to the data files context menu and make it the default "Open" setting. This is a useful workaround to the problem of losing a linkage when the filename is changed by the user, which is the main drawback to the filename association, but other factors may make even this approach unworkable in practice.

Since most of those programs which don't support file types also don't have any mechanism to accept a filename as a parameter, starting the program from the file is going to provide no extra benefit to the user. If anything, adopting this approach could be counter-productive, since the user would be using the same technique for starting both OS/2 and DOS programs but would get completely different results.

6.3 Problem Determination and Resolution

Perhaps the most common source of problems associated with OS/2 V2.0 is the failure to run the "shutdown" program before powering off the machine. "Shutdown" closes the system down methodically, giving all running programs an opportunity to save their data. It also writes the contents of the disk cache to disk if the *lazy write* option was set in HPFS.

Therefore, not shutting the system down methodically can result in system and application data being corrupted on the disk. This is especially troubling if the OS2.INI file becomes corrupted, since this file contains pointers to all the objects used by the WPS.

6.3.1 Error Symptoms of a Malfunctioning Desktop

The following symptoms may help in diagnosing when something has gone wrong with the system configuration files:

- Extra printers/queues have been defined in the INI file, but there is no printer object on the desktop; printing doesn't work properly
- Multiple instances of the same object(s)
- Impossible to use mouse button 2 to get the desktop's context menu
- OS/2 won't fully boot; the initial OS/2 sign-on is displayed but, when the Presentation Manager is supposed to start, the system hangs
- LAN login hangs the system.

6.3.2 Shadow Copies of Programs

Program files should not be "shadowed" to a folder on the desktop. Instead, a new reference for that program should be created within that folder. This prevents the user from writing over the name of the executable file if he directly edits (Alt-MB1) the icon description. Consistently applying this technique will also reduce the number of cross references within the OS2.INI file and enhance the reliability of the WPS.

6.4 Backup and Restore with the Workplace Shell

The Workplace Shell stores a great deal of critical data in the OS/2 initialization file OS2.INI and in Extended Attributes associated with data files and directories. The OS2.INI file contains system details such as the pointers for all the abstract objects - shadows, program references, etc. A fuller discussion of what the Workplace Shell stores in OS2.INI may be found in Chapter 8, "Workplace Shell Implementation" on page 111.

If this information is lost or corrupted for any reason, the effect on the system can be very serious. Backing up only the contents of data files is, therefore, no longer sufficient while backing up OS2.INI has gained critical importance.

This section discusses approaches to back up and restore that will allow a user to recover from system failures in such a way that his desktop environment is not disrupted too badly. It also discusses some of the more popular back up utilities in terms of their suitability for backing up a system using the Workplace Shell.

6.4.1 Critical System Files

OS/2 V2.0 has a built-in mechanism for copying and restoring the three critical system files: CONFIG.SYS, OS2.INI and OS2SYS.INI.

During boot, if you press Alt-F1 before the CONFIG.SYS file is read (the best time is when disk access begins), the system will use neither the CONFIG.SYS file found in the root nor the OS2.INI and OS2SYS.INI files found in the \OS2 directory. These versions of the CONFIG.SYS and .INI files are renamed with a numeric extension such as CONFIG.003 or OS2.015.

The system then replaces these files with versions of the CONFIG.SYS and the .INI files that are stored in the \OS2\INSTALL subdirectory. A message informs the user of what occurred and the system continues its IPL.

This mechanism works well for replacing existing copies of these files while preserving the old version in case the new one generates system errors. We have also discussed other approaches, below, which may offer additional flexibility.

6.4.2 How to Back Up OS2.INI

The OS2.INI file is kept open by the WPS at all times, so normal backup techniques cannot be used - the back up programs concerned will be denied access to the file as it is already open. One solution that we have found useful is to copy this file during the operating system boot **before the Workplace Shell has started**.

It is not sufficient to make only one copy; if you do that, and corrupt OS2.INI, the next time you boot the system your corrupted file will overwrite the backed up copy. It may even be that you do not know you have a problem until after the re-boot, by which time you will already have lost your backed up copy.

The solution to this is to make a series of generation backups of OS2.INI, by using XCOPY from within CONFIG.SYS and some COPYs from within STARTUP.CMD. Since these backup files are not used by the system they may be backed up to tape or another disk just like any other data files. Although the

critical WPS data is held in OS2.INI, it is worth backing up OS2SYS.INI in the same way.

This is illustrated in Figure 28 and Figure 29.

```
RUN=C:\OS2\XCOPY.EXE C:\OS2\OS2*.INI C:\OS2\INSTALL
```

Figure 28. Starting XCOPY From the First Line in CONFIG.SYS to Back Up the INI Files

```
REM *** Build Backup History ***
C:
CD \OS2\INSTALL
COPY OS2.3 OS2.OLD
COPY OS2.2 OS2.3
COPY OS2.1 OS2.2
COPY OS2.INI OS2.1
COPY OS2SYS.3 OS2SYS.OLD
COPY OS2SYS.2 OS2SYS.3
COPY OS2SYS.1 OS2SYS.2
COPY OS2SYS.INI OS2SYS.1
CD\
REM *** That's all folks ... ***
```

Figure 29. Building Back Up History of the INI Files from STARTUP.CMD

This scheme keeps five versions of the INI files on disk, .OLD being the oldest. If something happens to an INI file you still have a chance of reverting to a previous version. The space occupied by the backups depends on the size of your INI files and the number of cascaded copies you make; from 500 KB to several MB of disk space can be used up. You must make your own judgement as to the number of generations to keep, based on the size of the files concerned and the available disk space.

6.4.3 Restoring a Backup Version of OS2.INI

If you should be so unfortunate as to lose or corrupt the OS2.INI file, you will want to restore it from the most recent, clean, backup copy you have. This will normally be the one that was copied when you last booted the system.

Since OS2.INI is kept open at all times by the Workplace Shell, you cannot simply copy the backup over the current file. The two approaches to resolving this are outlined below.

6.4.3.1 Reboot from Diskette

This procedure will only recover your system to the point at which it was previously saved. Any objects and folders that you added since that point will be lost, as will any colors and desktop settings which you altered.

Booting from diskette to restore the OS2.INI file requires the following steps:

Step 1. Obtain the OS/2 V2.0 "Installation" diskette and diskette 1

Step 2. Insert the OS/2 V2.0 "Installation" diskette and reboot

Step 3. When prompted, insert diskette 1 and press enter. Wait for the first Install panel and press Escape for an OS/2 command prompt

Step 4. Copy your saved INI files into the bootup drive and directory:

```
COPY  A:\*.INI  C:\OS2
```

Step 5. Remove the diskette and reboot.

6.4.3.2 System Install from Alt-F1

The Alt-F1 keystroke combination, described in 6.4.1, "Critical System Files" on page 74 will copy the CONFIG.SYS, OS2.INI and OS2SYS.INI files from the OS2\INSTALL directory into the appropriate directories before reading those files. A fuller discussion of this technique may be found in *OS/2 Version 2.0 - Volume 1: Control Program*.

6.4.3.3 What is the Effect of Restoring a Back-level OS2.INI?

If you have to restore an old OS2.INI to an otherwise intact system, there may be conflicts between the contents of OS2.INI and the files, directories and EAs on the disks. The relationships between these are discussed more fully in Chapter 8, "Workplace Shell Implementation."

The degree of problems caused will depend on how many program files have been copied and how many shadow copies have been made since the date that the OS2.INI file was copied. This is because these objects are stored in the OS2.INI file and so will be lost when it is replaced.

The usual problem that ensues is that a folder will have a pointer to that program or copy in its directory EA, but now the pointer no longer exists. The solution is easy; perform a refresh on the folder, then copy the program or file shadow again.

6.4.4 Backup Programs

The backup programs PMTAPE and SY-TOS Plus** for OS/2 are capable of backing up the main system files (CONFIG.SYS, OS2.INI and OS2SYS.INI) as well as the directories, files and their associated EAs.

6.5 Using the Workplace Shell in a LAN Environment

The LAN independent shell in OS/2 V2.0 is an integral part of the WPS. If you have one or more requesters started in your CONFIG.SYS file, an icon labelled *Network* appears on the desktop. Opening the *Network* folder shows an icon for each network type. A user can move or shadow the *Network* folder to any other folder.

6.5.1 Organization of a LAN Workplace

The distribution of task-oriented resources between server and workstation is largely a matter of security, licensing, performance and the hardware capabilities. The planning and set up of a user environment should also consider issues such as LAN availability.

In a stable environment with file mirroring almost all data and programs could be stored on, and accessed from, a LAN server. The integration of the Workplace Shell and LAN permits the following combinations:

- Common programs and data can be stored on the LAN server
- Programs and data limited to certain users can be placed in dedicated folders on the LAN server

- Highly important programs, which must be available to a user even if the LAN is not running, may reside on the workstation
- Shadow copies of LAN data files can be set up in local folders within the desktop structure.

As a network is hierarchically organized it may seem to make sense to focus on the LAN servers for the placement of folders and data objects. However, using the *Network* and *LAN Server* folders as the standard way of providing access to programs and data would be inconsistent with the way the user accesses local resources.

This would mean that he would have to work his way through the hierarchy each time he needed access to LAN-resident resources. For this reason, a better approach would be to use shadow copies of the resources he wants to use from the network directories.

Several different approaches to arranging local and remote folders are possible. The possible combinations of folders and files are:

- Shadow folder with shadow files
- Shadow folder with local files
- Shadow folder with local and shadow files
- Local folder with local files
- Local folder with shadow files
- Local folder with local and shadow files.

In all cases, we assume that programs are stored remotely on the LAN server and that only files are displayed in folders, not programs.

6.5.1.1 Shadow Folders

The shadow folder is a pointer to the real folder (and associated directory) on the LAN Server. Any objects in the LAN folder will also be shadow copied into the shadow LAN folder. This approach allows the directory to be maintained by an administrator so the user can concentrate on the tasks he is paid to perform. It ensures that sensitive data can be secured and backed up at a central point.

The shadow LAN folder has some unique behaviors. It will not open if all the real objects reside on the LAN and the user has not logged on to the server. This is because the logical drive on the LAN cannot be accessed by the folder; this can be seen in the *File* page of the folder *Settings* view.

However, this situation is modified where local objects have been placed in the shadow LAN folder. In this case the folder can be opened but if the user then tries to access a shadow object before "login," the Workplace Shell issues a warning.

The following sequence is used to set up a shadow LAN folder:

Step 1. Create a shadow copy for the user's "home" folder from the LAN

Step 2. Put any local, real objects in the shadow LAN folder

Step 3. Create shadow copies of the local data file objects in the shadow LAN folder

Step 4. Create new program references for local programs in the shadow LAN folder

Step 5. Create new program references for remote programs in the shadow LAN folder.

Program files should not be "shadowed" to the local folder. Instead, a new program reference for the program should be created within that folder. See 6.3.2, "Shadow Copies of Programs" on page 73 for more information.

There are some differences in the usage of a shadow LAN folder and a local folder which may seem confusing to the inexperienced user. For example, if the user has a shadow copy of a LAN folder and then tries to make a shadow copy of a file from the LAN folder into that shadow folder, the WPS will, correctly, not allow it.

Another example would be where the user wants to "logout" but one of the objects in the shadow folder or the *LAN Server* folder is still in use. In this event, the Workplace Shell will report an *active connection* and refuse to close.

There are also disadvantages to using shadow copies of either a local or a LAN folder. For instance, the act of renaming a shadow folder by "direct editing" the icon description will result in the physical name being changed. This can cause problems on a shared directory where many users have Read/Write (RW) access. Under such circumstances it would be better to set up local folders with shadow copies of files, as described below.

6.5.1.2 Local Folder

Instead of a shadow copy of the LAN folder, a better approach is to use a local folder (or work area). All types of objects, both real (local) and shadow (local and remote), can be stored in a local folder. The advantage of this approach is added security against loss of data and more consistency in organizing the WPS by task.

The set up is completely transparent to the user and the behavior of shadow copies of LAN data files does not differ from 6.5.1.1, "Shadow Folders" on page 77.

The local folder resides in the local desktop structure. It therefore has a real name and the disk space needed depends on the number and type of objects stored in it.

To set up a local folder with LAN objects requires the following steps:

Step 1. Create a local folder

Step 2. Create shadow copies of the LAN data file objects in the local folder

Step 3. Create shadow copies of local data file objects in the local folder

Step 4. Create new program references for local programs in the local folder

Step 5. Create new program references for remote programs in the local folder.

Program files should not be "shadowed" to the local folder. Instead, a new program reference for the program should be created within that folder. See 6.3.2, "Shadow Copies of Programs" on page 73 for more information.

6.6 Workplace Shell Performance

Since users can open as many programs as they like, this is a potential cause of performance problems. In addition, since the WPS will restart any programs which were left running at "shutdown," we can conceive of a situation where performance would slowly degenerate over some time, with the operating system starting more and more programs each day, even if the user didn't need them all.

There are two techniques which can help prevent these problems. The first one is to use work areas instead of folders, then encourage users to close a folder when they have finished with it instead of minimizing it. When a work area is closed, all the programs in it are also closed, so this helps free resources for the next program(s) the user has to run.

The second technique is described in 6.8.1, "Prevent Programs Restarting at IPL" on page 80, where any previously running programs are prevented from restarting when the system is restarted. A REXX program is described which performs this.

In addition, some WPS functions are inherently slower than others. For example, opening a folder with a tree view is slower than opening it with an icon view. Users quickly learn to use the WPS functions in the most appropriate way.

6.7 Training Users to Use the Workplace Shell

A good understanding of the basic concepts behind the workplace environment will lead to greater user satisfaction and should help reduce their need for support.

It is important that users understand not just the techniques used to interact with the Workplace Shell, but also the objects that the WPS uses. They may think that, since they make a copy of a file using Ctrl-drag but a shadow copy using Ctrl-Shift-drag, then Ctrl-dragging a file will always make a real copy, no matter what kind of object the source is. Of course, that's not how the WPS works; if you copy a shadow, you get another shadow.

Shadows can cause other problems for the inexperienced user. For example, if you have a user who wants to copy a file to a diskette, they will not differentiate between the real file and a shadow, and will therefore fail to understand why the WPS won't let them drag that shadow onto the diskette icon. In general, it might be better to remove the diskette icon from the desktop and let users work with the *Drives* folder for all their file system interactions.

6.7.1 Training and Desktop Configuration

An important point to note here is that training for the WPS depends on what objects are put onto their desktop; the greater the variety of objects, the more the user has to know. Approaches to configuring the desktop are discussed in 6.8, "Utilities for the Workplace Shell" on page 80.

Restricting the range of objects to devices, such as printers and shredders, folders and data files is probably the best approach; it is very consistent and requires the user to learn the least number of techniques. This is described in 6.9.1, "User Requirements" on page 85.

Users with previous experience of using DOS or OS/2 will wish to continue to use programs since that is how they already know how to work with a computer. While the WPS is more logical, consistent and simpler than its "program menu" predecessors, it is often difficult for some users to adjust to its new style of interaction. Under these circumstances, modifying the shell to look more like the PM shell in OS/2 Version 1.3 might be a better approach. 5.4, "Giving OS/2 V2.0 the Look and Feel of OS/2 Version 1.3" on page 64 provides more information on this topic.

6.8 Utilities for the Workplace Shell

This section discusses various techniques that may be applied to the WPS to tailor it to the needs of specific users or groups of users. In general, REXX programs are used here to illustrate the points, although for security and performance these might be rewritten as C programs before widespread distribution.

These procedures assume that SysLoadFuncs has already been loaded, as shown below:

```
/* Rexx program that uses RexxUtil functions */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs
```

Refer to Appendix A, "Using REXX in OS/2 V2.0" on page 131 for more details.

6.8.1 Prevent Programs Restarting at IPL

On shutdown, a number of programs may be active which will be reactivated at the next system IPL. A standard technique is available to override this. It is described in the "README" file in the root directory. Press and hold the left Ctrl, left Shift and F1 keys while the operating system is IPLing. This disables the automatic program startup feature of the desktop.

This may not be appropriate for certain classes of user, for example where several people share the system. These people would rather find the system in a standard state each morning, regardless of what the previous user had been running the day before. It is unlikely that an adminstrator in a corporate environment would want his users to have to learn the Ctrl-Shift-F1 sequence.

A simple way to automatically prevent programs being restarted after an IPL is to use a small REXX program to clear the *Startup* folder before the Workplace Shell is activated. If this program is invoked from the STARTUP.CMD file, it will run before the Workplace Shell is initialized to prevent previously running programs from being started.

This is illustrated in Figure 30, below:

```
/* Clear startup programs from OS2.INI */
call RxFuncAdd 'SysIni','RexxUtil','SysIni'
call SysIni,'PM_WorkPlace:Restart','DELETE:'
call SysIni,'FolderworkareaRunningObjects','DELETE:'
```

Figure 30. A REXX Procedure To Prevent Programs Restarting

6.8.2 File Transfer to a Host Session

The following procedure is useful for uploading files from the PS/2 to a host system. Both methods hide file transfer programs from the user and thus helps him maintain a consistent way of working with the WPS.

Two alternative approaches are used; the first creates an icon onto which the user can drop the file to be transferred, the second simply adds a file transfer command to the pop-up menu.

6.8.2.1 File Transfer Icon

Step 1. Create a folder to hold the files that are to be uploaded

Step 2. Copy the CMD file in Figure 31 into that folder

Step 3. Create a program reference by pulling a program template into your folder

Step 4. Open a *Settings* view for the program reference

Step 5. Put your CMD file path and filename in the Physical Name field

Step 6. Put %* in the Parameters field

Step 7. Close the *Settings* view.

The following REXX procedure is called when an object is dropped on the program reference icon that was created above. The name of the object is passed to the procedure which then eliminates the path information from the object. A target file type is set up and an appropriate send option (ASCII or binary) is determined. Then the Communications Manager SEND command is used to transfer the file.

```
/*put a file on the host*/
'@echo off'
arg file
lastdot=lastpos('.',file)
lastbslash=lastpos('\',file)
ext=substr(file,lastdot+1) fn=substr(file,lastbslash+1,
lastdot-lastbslash-1)
select
when ext='SCR'  then ft='SCRIPT'
when ext='TXT'  then ft='TEXT'
when ext='PSE'  then ft='PSEBIN'
when ext='BMP'  then ft='BMPBIN'
otherwise  ft=ext
end
select
when ext='SCR'  then opt=asc
when ext='TXT'  then opt=asc
when ext='CMD'  then opt=asc
when fn='CONFIG' & ext='SYS'  then opt=asc
when ext='BAT'  then opt=asc
when ext='RC'  then opt=asc
otherwise  opt='(RECFM V'
end
if opt=asc then
opt='(ASCII CRLF RECFM V LRECL 255' 'SEND' file fn ft 'A' opt
'exit'
```

Figure 31. REXX Procedure for Host Upload

This REXX procedure does not take care of all possible problems. For instance, if the folder is on the desktop, a filename is created with the desktop folder name. As that name contains blanks, it needs to be enclosed in quotes, which is not done here. Therefore the upload folder needs to be in some other folder. However, the procedure shows how a simple REXX procedure can help create an object-oriented environment for a user.

6.8.2.2 File Transfer from a Pop-up Menu

For an alternative to the file upload icon, you can:

Step 1. Create a folder to hold program references

Step 2. Peel a program icon off the program template in Templates and drag it to this new folder

Step 3. Enter the path and filename of your file upload utility

Step 4. Enter the following for parameters: %* h[Enter host session letter (a,b,c,d) >]:

Step 5. Set the working directory to the path where your file upload utility resides

Step 6. Click on the General tab and change the name to "Upload to Host"

Step 7. Click on Associations and type an asterisk in the Names field, then click on the "Add" push button

Step 8. Close the Settings window.

Now, when you click on any file in your system with the right mouse button, you can click on the arrow to the right of "Open" and you will be offered "Upload to Host" as one of the programs that may be executed against it.

6.8.3 Limiting a User's Access to Settings

This requires a Workplace Shell program to be written. *OS/2 Version 2.0 - Volume 4: Application Development* provides some information on subclassing WPS objects which will be needed to accomplish this. The general approach is to subclass the object class and create a new one which doesn't allow access to settings, then remove the "Settings" choice from the context menu.

6.8.4 Creating and Populating Folders

One of the things an administrator will commonly wish to do is to to create and populate new folders for a user. The process is straightforward and can be automated using REXX programs. Several examples are provided below. The REXX commands used are explained in Appendix A, "Using REXX in OS/2 V2.0" on page 131.

The process involves:

Step 1. Creating a folder, which in turn will create a directory within the desktop structure

Step 2. Copying data files into it (optional)

Step 3. Creating new program objects within it.

For example, if you wanted to create a new folder on the desktop and call it *MyFolder*, use the following commands:

```
RetCode = SysCreateObject( "WPFolder",,
                           "MyFolder",,
                           "<WP_DESKTOP>",,
                           "OBJECTID=<MYFOLDER>")

if RetCode then
   say 'Folder Object created'
else do
   say 'Error creating object'
   exit(1)
   end
```

Figure 32. REXX Procedure to Create a New Folder

To add an editor to the folder, *MyFolder*, that was just created, use the following procedure:

```
RetCode = SysCreateObject( "WPProgram",,
                           "Editor",,
                           "<MYFOLDER>",,
                           "PROGTYPE=PM;EXENAME=C:\OS2\E.EXE;")

if RetCode then
   say 'Program Object created'
else do
   say 'Error creating object'
   exit(1)
   end
```

Figure 33. REXX Procedure to Add a Program to a Folder

The above examples show how to add instances of existing classes (Folder and Program), but you may also want to add new classes which you have created. You must first register a class with WPS before you can create an instance of it.

The example below shows how to register a *Password* folder. The design and coding of this folder is described in *OS/2 Version 2.0 - Volume 4: Application Development*. To register this new folder class:

```
RetCode = SysRegisterObjectClass( "PWFolder", "pwfolder")

if RetCode then
   say 'PWFolder Class registered'
else do
   say 'Error PWFolder Class failed to register'
   exit(1)
   end
```

Figure 34. REXX Procedure to Register a New WPS Class

To "deregister" the class when you want to remove it from the WPS classes, use this procedure:

```
RetCode = SysDeregisterObjectClass( "PWFolder");

if RetCode then
    say 'Uninstall successfully completed for PWFolder class'

say 'Re-boot NOW in order to release DLL'
```

Figure 35. REXX Procedure to Deregister a WPS Class

6.8.5 Adding New File Types

The following REXX batch file presents a fully supported method of adding types. It creates a new program reference plus the types associated with that program reference. If the specified types don't exist then they are added to OS2.INI. Afterwards, delete the program reference and the types will remain.

```
/* */
call RxFuncAdd "SysLoadFuncs", "RexxUtil", "SysLoadFuncs"
call SysLoadFuncs
call SysCreateObject "WPProgram", "Title", "<WP_DESKTOP>",
                     "EXENAME=EPM.EXE;ASSOCTYPE=Type 1, Type 2, Type 3,,"
```

Note that the call to SysCreateObject should only exist on one line in the batch file, not on 2 lines as shown above. The two commas after the "Type 3" are intentional.

6.8.6 Removing WPS Objects

In many cases, administrators will not want users to have access to all the objects provided with the Workplace Shell. One permanent way to remove objects is to create a custom OS2.INI file.

For example, to remove the shredder from the desktop, edit the "INI.RC" file in the "\OS2" directory and remove the shredder line:

```
"PM_InstallObject" "Shredder;WPShredder;<WP_DESKTOP>" "ICONPOS=908;OBJECTID=<WP_SHRED>"
```

Then make a new OS2.INI file by typing:

```
MAKINI NEW.INI INI.RC.
```

Replace OS2.INI with the contents of NEW.INI using the techniques described in 6.4.3, "Restoring a Backup Version of OS2.INI" on page 75.

6.9 Customizing OS/2 V2.0 for the Inexperienced User

The WPS presents a range of objects that must cater to the requirements of a wide variety of users. For many of these users this choice can be confusing and can, in some cases, reduce the usability of the WPS.

There is, therefore, a real need to provide a restricted range of objects which meet the exact needs of the user. While the user may have many objects, he will only want to use a small number of object types. The WPS allows him to complete any task by using only three types of objects:

- Containers (folders and work areas)
- Data files
- Devices (shredder and printers).

The user need not see, or understand, the concept of a "program" to be able to work with a file, thanks to the associations between data files and programs. See 8.5.1.1, "Running Programs" on page 127 and 5.3.5, "Associating an Object with a Program" on page 62 for more information on this topic.

Note that placing program icons directly in folders on the desktop can cause problems for the Workplace Shell as well as being inconsistent with the object-oriented approach we are trying to adopt. If the user makes a shadow copy of the program icon to another folder, then direct edits (Alt-MB1) the icon description, he can change the program executable name. The correct technique is described in 6.3.2, "Shadow Copies of Programs" on page 73. However, since this technique requires the user to know a lot more about the way the system works, it is probably better to simply eliminate this potential source of error by only using data file objects.

OS/2 Version 2.0 also provides new ways to pass information between programs running in different environments to provide as completely integrated a work space as possible. This saves time and prevents the user from introducing errors through rekeying data.

In addition, by removing extraneous objects, the user gains improved consistency of operation and thus enhanced usability. For example, double clicking on any object now opens a window which presents the contents of that object. This perspective is valid for data files, folders and devices, but not for programs.

This kind of consistency makes the system easier to learn for inexperienced users and encourages them to explore the capabilities of the system. An example of how to set up such a system is given in 6.9.1, "User Requirements."

6.9.1 User Requirements

This section describes the process of defining the OS/2 Version 2.0 requirements for a typical, inexperienced user, selecting the components and setting up the system. In this case, our user will be running on a stand-alone system. LAN considerations are discussed in 6.5, "Using the Workplace Shell in a LAN Environment" on page 76.

For instance, a user has a range of tasks to complete as part of his normal job. These include:

- Preparing quotations
- Entering orders
- Creating invoices
- Book-keeping
- General correspondence.

We will examine the task of creating customer quotations in some detail to illustrate how we would set up the system for any task.

Creating quotations involves:

1. Calculating a price for the goods or services requested

2. Filling these into a quotations document
3. Creating a cover letter
4. Printing the documents
5. Filing the documents.

Each task is represented by a work area. The reason we use a work area is that when it is closed it closes all the programs which have been opened from within it. Since memory and disk space may be restricted on our system, leaving programs open could cause us performance problems. Closing them when we finish one task and switch to another should help prevent that problem.

Each task involves other subtasks. For example, calculating the price might involve looking back at the outcome of previous quotations or checking to see whether the customer was traditionally a late payer (in which case a penalty charge might be levied within the price quoted).

When we look at the information required we see that we need the same information in several of the subtasks. The customer name and address would be needed in both the quotation document (which includes the terms and conditions which apply to the quotation) and the covering letter.

From the activities above, we can see that we need the following programs:

- A spreadsheet to calculate prices
- A word processor to produce and print the documents
- A database to store names and addresses
- A folder to store the completed files in.

We decided to use the following programs to illustrate a possible mix of operating systems and show what degree of interaction is possible under OS/2 Version 2.0.

- LOTUS** 1-2-3** /G for OS/2
- WordPerfect** for Windows**
- dBase IV** for DOS.

6.9.2 Operating System Set Up

Our objective was to provide a working system on an IBM PS/2* model 55-060 with 8 MB of memory. For performance and cost-effectiveness, we replaced the standard planar memory with 4 MB Single In line Memory Modules (SIMMs). The system is partitioned as follows:

C: Operating system partition - 23MB available

D: Applications partition - 35MB available.

The partition size for the operating system was chosen on the basis of requiring approximately 18 MB of disk storage for our operating system, plus 5 MB to allow us to add new features in the future. This will let us install new versions of the operating system, which might require reformatting the partition, without impacting our applications and data.

The desktop, spooler and swapper files were moved to drive D:. This was done as follows:

Swapper This was changed after the base installation, before completing the selective install.

Desktop This was performed after the installation. We used the *Drives* folder to move the entire desktop structure from the C: drive to the D: drive.

Spooler This was performed after the installation. We opened a "Settings" view of the *Spooler* folder and changed the path there.

For this user, we performed a selective install of the following OS/2 Version 2.0 components:

CD-ROM Device Support	Not installed
Documentation	We chose not to install REXX or Tutorial Documentation
Fonts	We installed the three outline fonts plus System Monospaced
Optional System Utilities	We installed Backup, Restore, Recover Files and Picture Viewer
Tools and Games	Not installed
OS/2 DOS and Windows	Installed
HPFS	Installed
REXX	Not installed
Serial Device Support	Installed
Serviceability Aids	Not installed
Optional Bit Maps	Not installed.

6.9.3 Setting up the Users Work Area

We created the *Quotations* work area by dragging a new folder from the folder template onto the desktop. We then renamed it, opened the settings view and checked the work area checkbox in the file page.

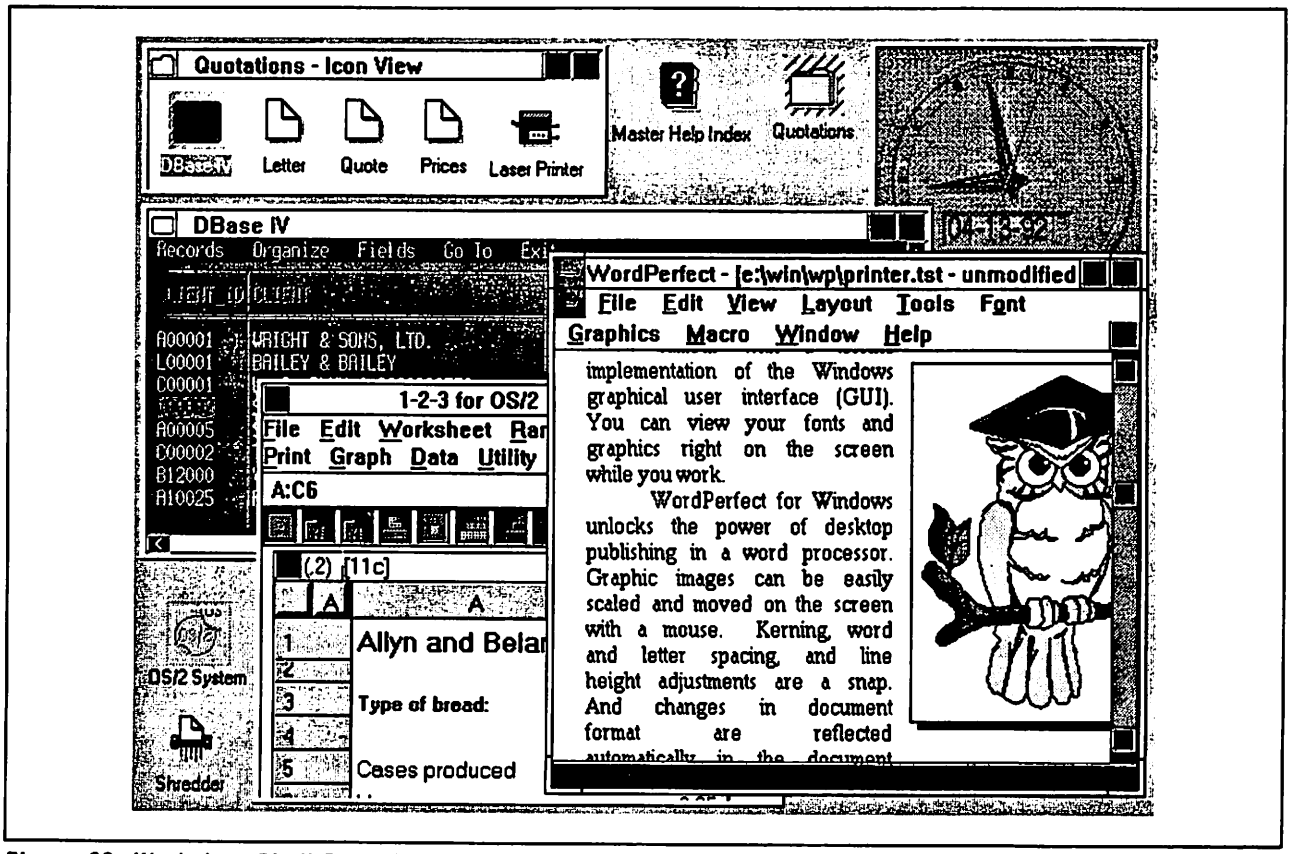

Figure 36. Workplace Shell Quotations Work Area

Within the work area we have the following templates:

- Quote.doc
- Letter.doc
- Prices.wg2.

Quote.doc and Letter.doc are WordPerfect documents, Prices.wg2 is a Lotus spreadsheet. They are all created in the following way:

1. Start the program, from the command prompt or by double clicking on its icon
2. Lay out the document or spreadsheet the way you want it
3. Save the file
4. Copy the file to the Quotations directory using *Drives*
5. Open the files *Settings* view and check the template checkbox on the "File" page
6. Close the *Settings* view.

The data file can be linked to the program in any of the ways described in 6.2.4, "Setting Up Programs in the Workplace Shell" on page 71. Since the DOS programs do not support the concept of file types, an association based on file extension was set up in each program.

The *Quotations* folder was created in the same way as the work area, that is, a new one was created by dragging it from the folder template and dropping it into the *Quotations* work area.

The user can now tear off a new "Prices" spreadsheet, rename it using some combination of the customer name and date, open it to fill in the necessary details, print it, then file it by dropping it in the *Quotations* folder.

When the user needs to switch to another task, such as creating invoices, he closes the *Quotations* folder. This stops any running programs, freeing the system memory and SWAPPER.DAT file. He then starts the next task by opening its folder.

6.10 Summary

The structure of the WPS is heavily dependent on the critical system files: CONFIG.SYS, OS2.INI and OS2SYS.INI. This chapter described several approaches to backing up and recovering these files.

The Workplace Shell is capable of being tailored to meet a variety of user environments. Restricting the functions provided to the inexperienced user results in a logical, consistent interface that is simple to learn and to use. The installation of such a user environment was described, together with some notes on the OS/2 V2.0 functions needed to support that user.

For the advanced user, OS/2 V2.0 provides a wealth of new techniques to help him become more productive. Several techniques are available to let the advanced user modify his environment, including adding programs to object menus. In addition, some REXX utilities have been provided to help the advanced user or administrator to modify the default operating characteristics of the WPS.

Installing and customizing stand-alone workstations is only part of the administrator's brief. Many factors must be considered, including disk partitioning, problem determination and overall system performance. LAN integration is a major enhancement to OS/2 V2.0 but this, too, must be carefully planned to ensure that the environment created really does meet the needs of the users.

Chapter 7. Presentation Manager and Workplace Shell Application Development

OS/2 Version 2.0 introduces a new choice for application developers: whether to develop their applications using traditional PM programming techniques or to use the System Object Model (SOM) interfaces and the Workplace Shell class hierarchy to develop fully-integrated applications.

This chapter provides an overview of the two programming models, and discusses the extent to which it is possible to integrate a PM program into the Workplace Shell environment.

For more detailed information on programming for Presentation Manager and for the Workplace Shell, see *OS/2 Version 2.0 - Volume 4: Application Development.*

7.1 The Presentation Manager Application Model

The conceptual model upon which a Presentation Manager application is based differs somewhat from "conventional" application models. The components of a conventional application communicate with one another via function calls and pass information in the form of parameters. The components of a Presentation Manager application are called windows, and communicate using messages which are transmitted between windows by Presentation Manager on the application's behalf.

This section will examine the conceptual application model implemented by Presentation Manager. The intent of this section is to provide the reader with an introduction only. A more detailed examination of the application model from a programmer's viewpoint is given in *OS/2 Version 2.0 - Volume 4: Application Development.*

7.1.1 Windows

Presentation Manager applications are based upon the concept of **windows**. A window typically represents some object, such as a file or document, a device or a data record, upon which the application will operate. The user interacts with the window to manipulate the object.

A window appears to the user as a rectangular area on the screen, which may be moved and re-sized by the user with either the keyboard or mouse. From an application viewpoint, however, the concept of a window is far more powerful than this. Windows may be of two basic types:

- **Display windows** have a visual manifestation represented by a rectangular area on the screen; in this case, the window represents a view of a conceptual display surface known as a **presentation space**, which is the object upon which the window operates. This view may be full or partial, depending upon the current size of the window, and the size of the presentation space. See 7.1.3, "Presentation Spaces and Device Contexts" on page 97 for more information on presentation spaces.
- **Object windows** have no visual manifestation, and are merely addresses or "handles" to which messages may be directed. An object window is typi-

cally associated with an object such as a file or database, and is used to access this object and perform actions upon it.

Windows respond to events which are communicated to them by way of **messages**. Messages may originate from Presentation Manager as a result of user interaction, or from other windows in the system. Messages may be of many different types; Presentation Manager defines a number of **message classes**, and an application may define its own message classes for communication between its own windows. Messages are routed between windows by Presentation Manager on behalf of the applications, and are discussed in greater detail in 7.1.2, "Messages" on page 94.

The basic structure of a Presentation Manager application is therefore that of a group of windows, communicating with one another by way of messages. This is illustrated in Figure 37.

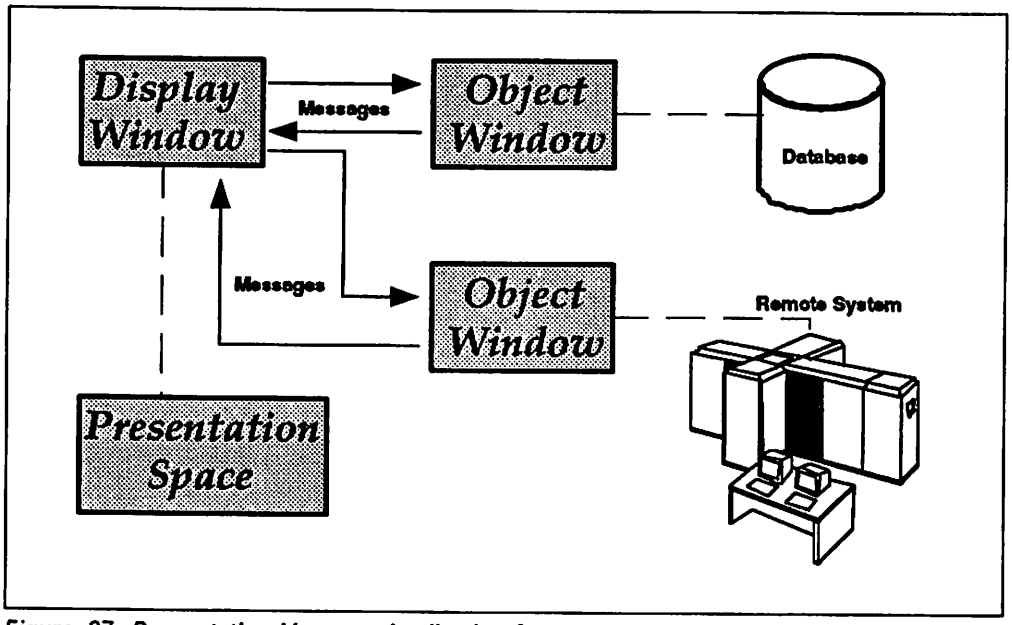

Figure 37. Presentation Manager Application Structure

The behavior of a window in response to messages directed to it is determined by its **window procedure**, which determines the processing performed by the window in response to each message it receives. Windows with similar characteristics are grouped into a **window class**, and share a window procedure.

Note that windows are a finite operating system resource. Under OS/2 Version 1.1 and Version 1.2, the maximum number of windows which could be created in the system was approximately 1200. Under OS/2 Version 1.3 this limit was increased to approximately 10000; this increased limit also applies to OS/2 Version 2.0.

Other Presentation Manager objects such as presentation spaces and device contexts also consume storage for control information. Under OS/2 Version 2.0, the total limit for all graphics resources used by Presentation Manager is approximately 64000.

Note also that when large numbers of windows are created in the system, Presentation Manager uses significant processor resource to handle message routing, clipping etc. Some degradation in overall system performance may

therefore be experienced when running with an extremely large number of windows open.

7.1.1.1 Window Classes

Each window belongs to a window class. The window class determines a number of properties of the window, including its window procedure, which in turn determines the behavior of the window in response to the messages it receives. A window class is registered with Presentation Manager, and may be defined in one of two ways:

- *Public*, in which case the window class is registered automatically at system initialization, and may be used by any application in the system.

  A number of window classes, such as the frame window and control windows, are publicly defined by Presentation Manager, and are hence available for use by applications.

- *Private*, in which case an application must register the window class explicitly during its own initialization. Windows of this class may then be used only by that application.

Each window within a class is said to be an instance of that class. Multiple windows of the same class may exist in the system at the same time, controlled by the same or different applications.

7.1.1.2 Window Procedures

Each window in the system has a window procedure, which defines the processing performed by the window for each message it receives. Such processing may include general application logic, I/O operations or communication with other windows. The window procedure is associated with an entire window class rather than an individual instance of the class, and is defined when the class is registered to Presentation Manager.

The window procedure must be provided by the application which registers the window class. Presentation Manager provides its own window procedures for its publicly defined window classes. Applications which register their own window classes must provide a window procedure for each class.

The window procedure is invoked by Presentation Manager on the application's behalf, in response to any message directed to that window, either from Presentation Manager itself or from another window in the system. The window procedure determines the class of the message, and takes appropriate action. Since there are an extremely large number of messages which may be passed to a window, the window procedure need only process those messages which are required to carry out the window's functions; Presentation Manager provides default processing for all system-defined message classes, and application-defined message classes not processed by the window procedure are simply ignored.

Window procedures are re-entrant, and multiple instances of the same window class share the same memory-resident copy of the window procedure. Hence any local data defined by the window procedure is also shared by each window in the class. To allow each window to keep its own separate data areas, Presentation Manager allows an application to define an area known as the **window words**, which is unique to each individual window, and is maintained with Presentation Manager's own control block for that window. Window words are

normally used to store a pointer to a memory object which is dynamically allocated when the window is created, and used to store instance-specific data.

7.1.2 Messages

Messages are the mechanism by which windows communicate with one another and are notified of system- or user-initiated events by Presentation Manager. In a typical Presentation Manager application, all interaction between the user and windows, or between one window and another, takes place by way of messages. Messages may be of three types:

User-initiated The message is generated as the direct result of the user selecting a menu bar item, pressing a key on the keyboard, etc.

Application-initiated The message is generated by one window within the application for the communication of an event to another window.

System-initiated The message is generated by Presentation Manager as the indirect result of user action such as the window being resized or moved, or as the result of a system event such as window being created or destroyed.

Since almost every event which occurs within the system results in a message being generated, a Presentation Manager application has great freedom in the way in which it processes such events.

7.1.2.1 Message Queues

Whenever an event occurs within the system, such as the user pressing a key or selecting an item from a menu, a window being created or destroyed, or one window communicating with another, a message is generated and placed on a system message queue. Such messages are placed on the queue in the order they originated.

Each application has its own message queue, and each thread within a multi-threaded application may also possess its own message queue. This queue is explicitly created by the thread, via a function call to Presentation Manager, at the time the thread is initialized. (The term "thread" will be used herein, since message queues are always created on a per-thread basis; the application's initial message queue is created by its primary thread.)

Note that a secondary thread in a multi-threaded application need only create a message queue if it will create one or more windows. A secondary thread which does not create windows does not require a message queue.

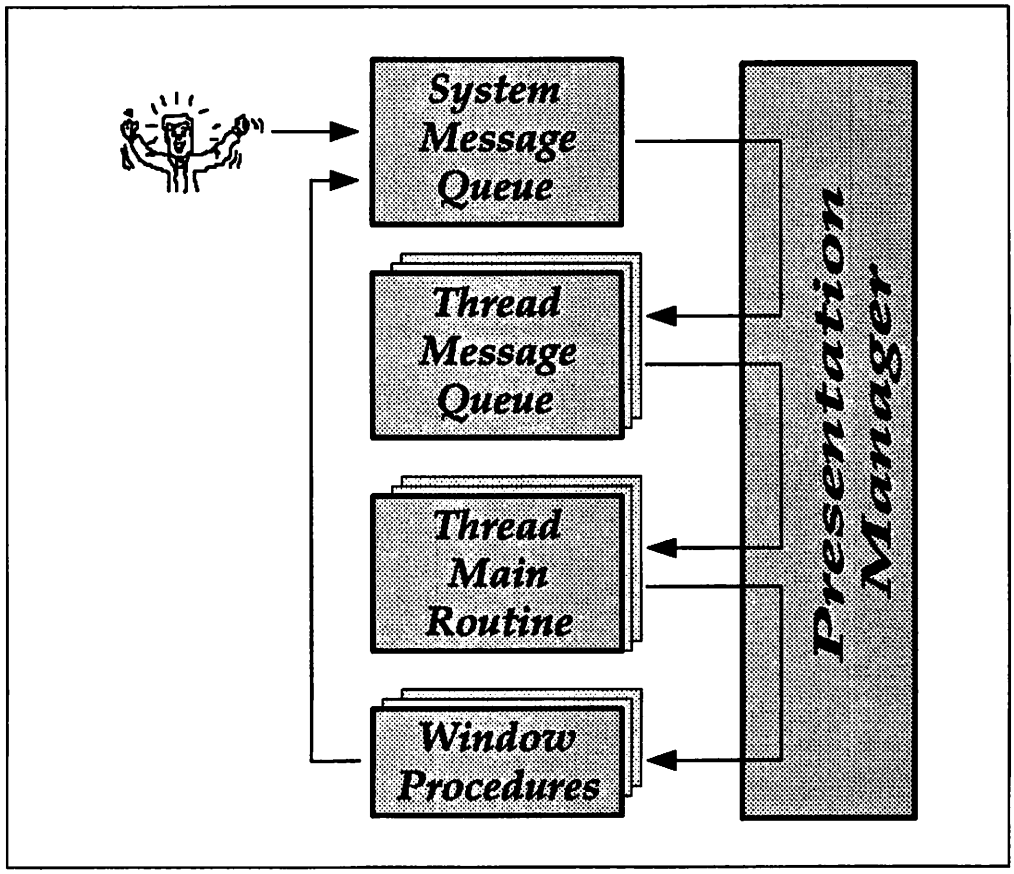

Figure 38. Message Queues

Presentation Manager periodically interrogates the system queue, removes each message and determines the window for which it is destined. It then places the message on the message queue belonging to the thread which created the window.

The thread's main routine, once initialization is complete, consists entirely of a message processing loop. The thread simply retrieves a message from the message queue via a function call to Presentation Manager, and requests Presentation Manager to pass the message to the appropriate window procedure, which is already known to Presentation Manager through the window class definition.

Presentation Manager then invokes the window procedure to process the message. When the processing is complete, the window procedure passes control back to Presentation Manager, which then returns control to the thread's main routine to obtain the next message.

If the thread's own message queue is empty when it requests the next message, Presentation Manager checks the system queue. *Note that this is the only time at which the system queue is accessed; hence no other window in the system can receive a message until the currently executing window procedure returns control to Presentation Manager.*

If a window procedure does not return from processing a message within a reasonable period of time, the user is effectively "locked out" of the system. In order to avoid such situations, applications which perform lengthy operations such as document formatting or remote system access, should be implemented

using multiple threads, thereby allowing the application's primary thread to continue execution and return control to Presentation Manager. This technique is discussed in *OS/2 Version 2.0 - Volume 4: Application Development.*

7.1.2.2 Message Classes

Messages are defined and identified using message classes. A message class is simply an integer used to identify the event which caused the message.

Message classes may be of two types:

- *System-defined* message classes are defined by Presentation Manager, and are used to indicate standard events such as a window being created or destroyed, sized or moved, or a key being pressed.
- *Application-defined* message classes are defined by an individual application, and are used to indicate events for communication between windows in that application.

Message classes are usually defined as integer constants. All system-defined message classes are defined in the header files which are shipped with the *IBM Developer's Toolkit for OS/2 2.0.* Application-defined message classes are usually defined in the application's own header files.

The structure of a message also includes two message parameters, which are 32-bit fields passed to the window procedure along with the message. These fields may contain additional information to aid in the window procedure's processing, or may contain pointers to memory objects which in turn contain such information.

7.1.2.3 Message Processing

Through its window procedures, a Presentation Manager application has the ability to process messages of any type or class, in the following ways:

- The window procedure may ignore the message, and simply leave it for Presentation Manager's own default window procedure, which will provide default processing for system-defined messages classes.
- The window procedure may explicitly process the message, using its own logic, calling subroutines and/or passing messages to other windows as necessary. This allows processing of application-defined message classes, and application-specific, "non-standard" processing of system-defined message classes.
- The window procedure may explicitly process the message and then pass the message on to Presentation Manager's default window procedure. This allows application-specific processing to be performed on system-defined message classes, in addition to the default processing.

Messages may also be processed in either of two modes:

- *Synchronous* processing occurs where the routine which passes the message (typically a window procedure or a subroutine invoked by a window procedure) waits until the message is passed to the target window and processed by its window procedure. Presentation Manager does not return control to the calling routine until the window procedure has completed its processing.

- *Asynchronous* processing occurs where the routine which passes the message waits only until the message is placed in the thread's message queue. Presentation Manager then returns control to the calling routine.

Presentation Manager also provides facilities which allow an application to broadcast messages to multiple windows with a single function call.

7.1.3 Presentation Spaces and Device Contexts

A display window provides a view into a conceptual display surface known as a presentation space. In terms of the above definition of a window, the presentation space is actually the object upon which the window and its window procedure operate. The contents of the presentation space represent application data such as a document or graphical image. The application places text and graphical items such as lines, arcs, and colors, in the presentation space by means of PM API functions (the *GPI* functions).

The window therefore provides the user with a view of the presentation space, using the screen. This view may show the entire presentation space or only a portion, depending on the size of the window and that of the presentation space. The size of the window is controlled by the user, although it is limited by the physical size and resolution of the screen. The size of the presentation space is controlled by the application and is limited by the amount of available memory which may be used to contain the presentation space.

A presentation space is usually created by a window procedure when a window is created, and is associated with a *device context* at that time.

A device context relates a presentation space to a physical device such as the screen or a printer, by converting the device-independent information stored in the presentation space to a device-dependent form that can be displayed on a particular device. If the contents of the presentation space must later be drawn on a device other than that for which the presentation space was initially created, the presentation space may simply be re-associated with a different device context. This may, for example, be done when a graphical picture in a window on the screen needs to be printed. The presentation space that was originally associated with a screen device context is re-associated with a printer device context.

7.1.4 Presentation Manager API Enhancements in OS/2 V2.0

The 32-bit Presentation Manager API remains largely unchanged in OS/2 V2.0, though there are a number of new functions, mostly to help the programmer rather than adding significant new function. For a complete review of the changes and enhancements, see *IBM OS/2 Version 2.0 Application Design Guide*, which includes a chapter comparing 16-bit and 32-bit OS/2 functions, including a section covering Presentation Manager. The following section is a summary of that information.

7.1.4.1 Functions Removed

A number of 16-bit PM functions are not included in the 32-bit set. In most cases these have been replaced by new 32-bit functions, or are no longer appropriate with the new shell. Areas affected are:

- Heap management functions
- Program list (group windows)

- Initialization file functions
- Window locking
- Window management.

7.1.4.2 Printing

All the printing functions, which previously had names with the prefix *DosPrint*, have been renamed to use the prefix *Spl*.

7.1.4.3 Workplace Shell

A number of new functions have been introduced giving PM programs an interface to the shell. These include, for example, the **WinRegisterObjectClass()** and **WinCreateObject()** functions, which allow a program to register and create Workplace Shell objects. The self-explanatory **WinShutDownSystem()** function provides a capability that was missing in previous releases.

7.1.4.4 Dynamic Data Facility

These new functions, which have the function name prefix *Ddf*, are intended to be used by programs that provide IPF text dynamically at run time. Such programs receive messages from IPF when they are to display their information, at which time they must build and display the text requested.

The DDF functions enable the program to format the text into paragraphs, lists and headings in such a way that it can easily be reformatted when the window is resized. Functions are provided to allow the creation of hypertext links, references to bitmap data, and to specify what sort of text formatting is required (left or right justification, for example).

7.1.4.5 Standard Font and File Dialogs

Many applications offer their users the option of loading and saving files from and to disk, and CUA defines the dialog that should be presented to the user so that he can select the appropriate disk, directory and file to use. The standard file dialog function, **WinFileDlg()**, implements this dialog with very little programming effort.

The standard font dialog, invoked by means of the **WinFontDlg()** function, does the same for those applications that allow font selection.

These dialogs, which are also discussed in Chapter 3, "New Presentation Manager Features," significantly improve programmer productivity when file or font selection facilities are required. They also ensure that the dialogs used by different applications present a totally consistent user interface.

7.1.4.6 New Window Classes

OS/2 V2.0 introduces several new control window classes, which are described in Chapter 3, "New Presentation Manager Features," along with some corresponding new messages.

7.1.4.7 Graphics Functions

A number of new GPI functions are provided, mostly related to the use of fonts and characters.

7.1.4.8 PM Helper Macros

A set of helper macros is provided that simplify interaction with some of the more commonly used control window classes. The use of the macros can reduce the need explicitly to send messages to such windows for simple tasks such as inserting an entry into a list box, or querying the state of a checkbox.

Being macros, these functions are expanded by the C pre-compiler into PM calls to send the relevant messages so, although they are coded as functions, no type-checking will be applied by the compiler to the arguments.

7.2 The Workplace Shell Application Model

The Workplace Shell provides an environment in which applications may be developed along fully object-oriented lines, integrating themselves seamlessly into the desktop environment. The techniques for developing such applications are discussed in this section; for more detailed information on this subject see *OS/2 Version 2.0 - Volume 4: Application Development.*

7.2.1 Workplace Shell Objects and Applications

The Workplace Shell provides a number of standard object classes such as folders and data files. The user performs his work by interacting with these objects using their context menus or direct manipulation.

In order to extend the range of tasks that a user can perform using the Workplace Shell, it is necessary to add new object classes to his desktop; typically these may be related to specific productivity tools, such as a *Spreadsheet* object, or to the user's own business, such as *Order Form*, *Parts Catalog*, or *Customer.*

In the ideal, purely object-oriented, user interface, there would no longer be anything that a user would recognize as a program - there would only be objects, all with their own unique behaviors and uses. As long as the user is provided with suitable tools (that is, object classes), he can work out how to accomplish any particular task without having to learn to use an application program specifically designed for that task. What we might loosely call a Workplace Shell application is really no more than a collection of Workplace Shell object classes.

In order for a newly developed object class to be used, it must be registered to the Workplace Shell. The classes are implemented in such a way that each class, or possibly several classes, is contained in a dynamic link library (DLL). Being in a DLL means that an object's methods can be called by Workplace Shell whenever necessary, and also that the code can be shared between multiple instances of the class. Registering a new class informs the Workplace Shell of the existence of the new class, and gives it the name of the DLL containing its methods.

The Workplace Shell itself, and all its classes including any that a user may develop, are written using the *System Object Model*, a language-independent environment for object-oriented programming. Anyone wishing to develop new Workplace Shell classes must therefore understand SOM, and also be familiar with the existing Workplace Shell classes, from which his new classes must inherit at least some of their behavior.

7.2.2 System Object Model

The System Object Model (SOM) is a language-independent specification for object-oriented programming (OOP). It consists of a set of utilities and interfaces for creating objects. This section provides a brief introduction to the System Object Model; for a more complete explanation, see *OS/2 Version 2.0 - Volume 4: Application Development*. That document also contains a fuller discussion of OOP concepts.

SOM implements all the essentials of OOP, including inheritance, encapsulation and polymorphism. Objects are organized into classes in a hierarchical manner and subclasses may inherit behaviors and characteristics from their parent (or *super*) classes.

SOM is language-independent. With suitable bindings any programming language may be used to implement the methods of an object class. For example, a class written in COBOL could then be used or even subclassed by another class written in Smalltalk/V**. However, so far the only language for which such bindings are available is C.

One of the great benefits of building WPS objects using SOM is that SOM implements the concept of *inheritance*. All objects are grouped into classes, and characteristics and behavior common to more than one class can be defined as methods in a superclass which are then inherited by all child (or *sub*) classes. Many common methods such as these are defined at the system level, in object classes supplied with OS/2 V2.0, and may be inherited by application-specific object classes.

An application developer may choose to enhance a system-supplied method: for example, providing a more advanced level of drag/drop functionality to a Workplace Shell object class. The developer may create a new object class as a subclass of the system-supplied object class. The subclass need override *only* those methods that are invoked in drag/drop operations. Other methods may simply be inherited from the super class.

It is important to understand that SOM is a general-purpose implementation of object-oriented programming, and may be used for any OOP programming under OS/2. On the other hand, the class hierarchy it provides is very limited, consisting as it does of only three classes, so a great deal of work would be required to implement classes if SOM were to be used for general-purpose object-oriented programming. In practice the only use to which most programmers will put the SOM is in developing the Workplace Shell objects that will form part of their applications.

7.2.2.1 OIDL and the SOM Compiler

Non-OOP languages such as C lack the language constructs for defining the classes, methods, instance data and so on that are needed when developing OOP objects. Even OOP languages which are designed for this can only do so for applications that are implemented entirely in one language. For these reasons, SOM provides its own language for defining classes, called *Object Interface Definition Language (OIDL)*. The details of a class, its instance variables, methods and the name of its superclass, are all defined using OIDL, and are placed in a file known as the *Class Definition File*.

The class definition file is used as input to the *SOM compiler*, which uses the information to generate several new files, such as C header files, a module defi-

nition file to be used when linking the object, and a skeleton C source file containing a prototype function for each method defined in the OIDL. The programmer inserts the application function he requires into this source file to implement the methods he wants, and then compiles and links the program in the usual way to produce DLL containing the object class.

A class definition file contains the following sections:

Include Section This section consists of a statement to include the class definition file of the parent class.

Parent Class Section This section defines certain basic information about a new class, such as its name and release level, and, by convention, comments that describe its position in the class hierarchy.

Release Order Section This section is used to allow a programmer to specify in which order the methods and public data of his class will be released. This can be important if the class is subsequently subclassed; without the release order having been specified, it could prove necessary to recompile the parent class in that case.

Metaclass Section This optional section is used to give information about the class's metaclass (that is, the class of which the new class is itself to be an instance).

Passthru Section This section allows a programmer to include some programming language code that will be placed by the SOM compiler into the language source file it generates. It is typically used for such things as *typdef* and *#define* statements.

Data Section This section specifies the instance variables to be used by the class.

Methods Section This section lists all the methods which are to be defined by the class, both those that are new to the class and those of its parent class that it will be overriding.

7.2.3 Using Workplace Shell Classes

The Workplace Shell introduces several new classes derived from the *SOMObject* class; for example, class definitions for a folder (*WPFolder*), a program reference (*WPProgram*) and a printer (*WPPrinter*). The class hierarchy of the classes used in the Workplace Shell is illustrated in Figure 39 on page 102.

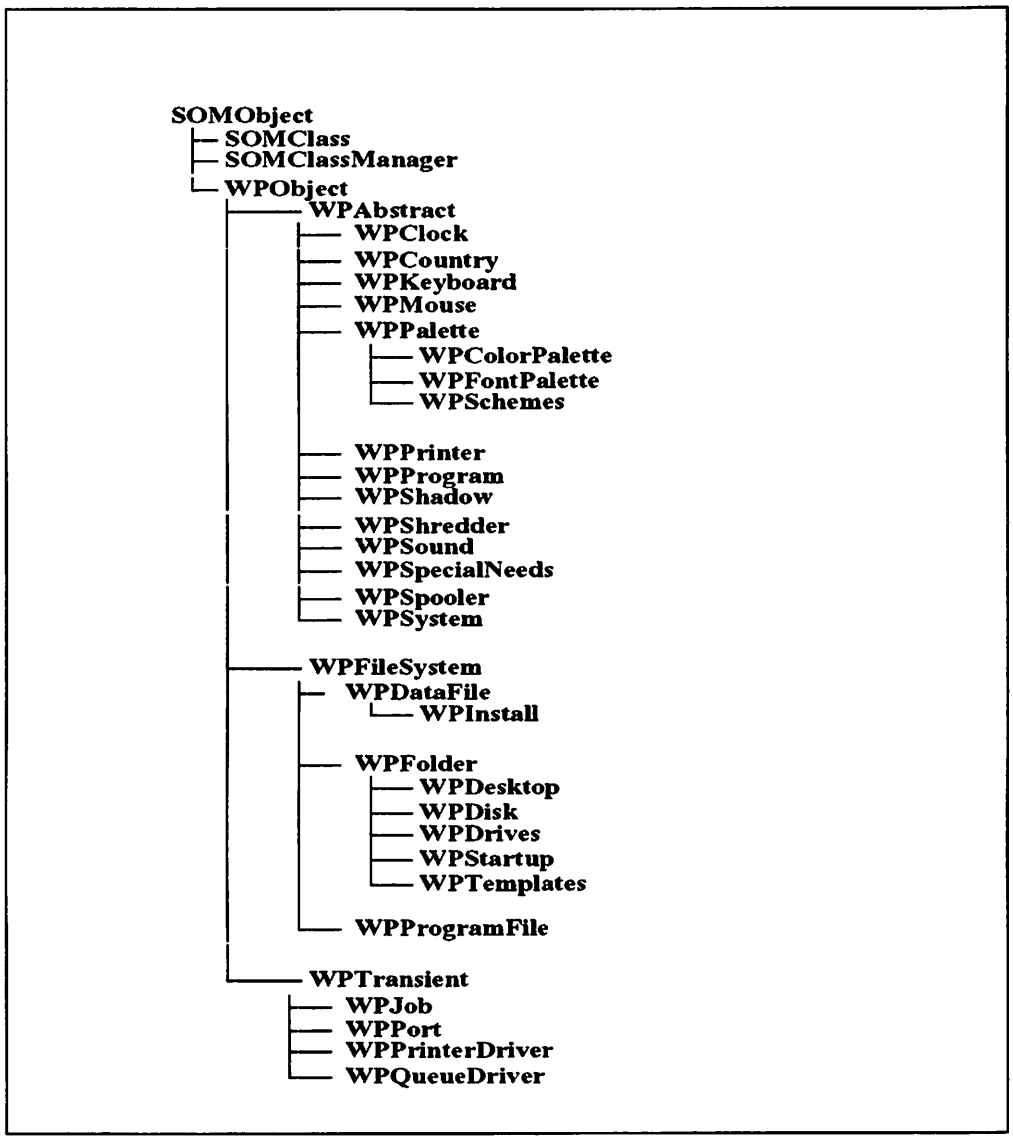

Figure 39. Workplace Shell Class Hierarchy

Notice that this hierarchy includes three so-called *base classes* from which all the remaining classes are derived; these base classes are also known as *base storage classes* because the fundamental difference between them, and therefore between all their derived classes, lies in the way they store their control data. For a fuller discussion of this see Chapter 8, "Workplace Shell Implementation."

If you want to implement a new WPS class, you must first decide which of the existing classes provides the most promising base from which to work - you have to subclass one of them. Early experience suggests that subclassing the existing base classes or their derivatives is usually the more practicable approach - implementing a new base class by subclassing WPObject may be desirable but should be regarded as a very major piece of work.

7.2.4 The Structure of a Workplace Shell Application

The Workplace Shell has been implemented in such a way that the whole shell and all its objects, be they system supplied or user developed, run as a single OS/2 process. This has some very significant implications:

- If an object abends for any reason, the whole shell and all its objects crash, disrupting the user's work, and possibly resulting in lost data.
- If a method of a Workplace Shell object takes too long to complete its processing, the whole user interface will be locked up until it completes.

  This is a problem familiar to PM programmers, and as with PM, the solution is to start a second thread for the long-running code. Typical situations where this problem can arise include retrieving data from a database or communicating with a host system.
- There is potential for data integrity problems, as it is possible for all objects to access each others' data; this is probably more of a problem in theory than in practice, since most accidental addressing errors will still result in "Trap" errors.

The recommended solution to these problems is to split an application into two parts: one part, consisting of one or more WPS objects running in the WPS process, the other part containing most of the application logic running as a separate OS/2 process. The two parts communicate using the interprocess communications (IPC) facilities of OS/2.

To avoid the problems above, it is best to put as little of the application as possible into the WPS objects themselves. Clearly some functions must be there - for example the code to create and handle an object's context menu, or to handle direct manipulation - and we have found it best also to code PM dialog and window creation and their dialog and window procedures within the WPS objects. Other application functions, such as database access, communications and calculation, should be put in a separate process.

This structure approximates to the Model-View design approach favored by object-oriented programmers. Our recommendation effectively places the *View* in the WPS process and the *Model* in a separate process.

The Workplace Shell documentation provides no guidance on good and bad ways of implementing an application using IPC between the shell process and the application processing thread. *OS/2 Version 2.0 - Volume 4: Application Development* includes a discussion of this and describes one approach that has been tried, along with programming examples.

7.3 Writing PM Applications to Work with the Workplace Shell

The Presentation Manager has been enhanced to include several new classes for use with the Workplace Shell, notably the *Container* and *Notebook* classes. The OS/2 V2.0 Presentation Manager would seem to have all the basic requirements for writing programs that have the capabilities of Workplace Shell applications, without programmers having to learn SOM or become familiar with the Workplace Shell class hierarchy.

The question that arises therefore is this: how well can a straightforward PM program that uses the new classes and the drag/drop protocols integrate itself into the WPS environment? This section helps to answer this.

7.3.1 The Workplace Shell and PM

The Workplace Shell along with all its registered classes, be they system-supplied or user-written, is itself just a Presentation Manager program. As such it should be able to cooperate quite well with other PM programs, as long as it conforms to the protocols laid down by PM, particularly for drag and drop.

The technical term for the drag/drop protocols is *Rendering Mechanisms*; the standard ones are defined in the *OS/2 2.0 Programming Guide Volume II*. The Workplace Shell implements the *Print* and the *OS/2 File* rendering mechanisms. These make it possible for a PM application to provide direct manipulation printing capabilities and, if required, file drag/drop facilities similar to those provided in the OS/2 Version 1.3 File Manager.

7.3.2 How Much Can You Do with PM?

You can emulate many Workplace Shell characteristics in a straightforward PM program. For example you can:

- Display container windows, which look very much like WPS folders, and populate them with icons that look superficially like WPS object icons.
- Implement context menus for both icons and windows.
- Provide drag/drop facilities for the items handled by the program. When these items represent files the standard OS/2 File rendering mechanism will allow them to be dragged in and out of WPS folders. When it makes sense to print these items, they can support the standard Print rendering mechanism, to allow printing by direct manipulation.

Although a PM program can participate to some extent with the Workplace Shell, there are still things the user may want to do that cannot easily be implemented without developing the application using SOM and WPS.

Consider a PM program that displays some icons in a container window representing customers of a particular branch of a company. The user sees these icons in a window that looks very much like a WPS folder. He may reasonably expect to be able to drag the icon of a customer in whom he is particularly interested onto the desktop or into another folder.

How can a PM program handle this? The Workplace Shell supports the OS/2 File rendering mechanism, but that is not really suitable for customers, whose details are more likely to share a database table than to occupy a separate file each. The PM program may allow the drag to take place, but cannot support any rendering mechanism that a WPS folder will understand, so the user sees the "inhibited" pointer while the customer icon is over the desktop and is not allowed to drop it.

This confuses the user, who expects to be able to put any object anywhere he chooses.

7.3.3 Migrating Existing Applications

Many OS/2 users will find themselves with existing PM applications that they have developed, and may wonder whether they can modify them to exploit the Workplace Shell environment. Ideally, one would like every application to be built as a collection of WPS objects, interacting with one another and with the standard WPS objects, but many users will decide that converting existing PM programs to this application model is not justified.

There are two approaches one may take to application migration: one is simply to make a few minor modifications so that the application works better in the new environment, the other is to convert the whole program to the WPS/SOM application model, carrying over as much code from the PM version as possible.

7.3.3.1 Minimal Changes

A particularly useful facility provided by the Workplace Shell is *Association*; this can be by file type, filename or extension. A program may be associated with one or more data files. The effect of this is that the program concerned automatically becomes one of the views available from the *Open* action on the context menu of any file with this type. Opening this view causes the associated program to be started with the filename as its first command line parameter.

Many programs that operate on files are written to expect a filename as their first parameter, so these programs can be used in the Workplace Shell environment in a way that is consistent with the object-oriented user interface - the user chooses the object he wants to work on and opens it; the window presented by the program can be thought of as a view of the object.

Any program that does not accept a filename as a command-line parameter can easily be modified to do so.

Other changes that could be considered:

- Change the way the program ends, so that by default it saves its data back into the file from which it read it. This is more consistent with the way a Workplace Shell object allows you to open a view, make changes, then close the view, without having explicitly to save the data first.
- Replace the menu bar with a context menu. This may or may not be desirable - menu bars are still a quite acceptable part of the user interface and are required by CUA 91, but are little used within Workplace Shell itself.
- Implement some simple direct manipulation - doing this for printing can be very straightforward, and makes the program fit much better into the Workplace Shell environment.

  Supporting drag/drop for files, even in a limited way, can be very helpful. As an illustration, consider the Enhanced Editor provided with OS/2. This is a standard PM program but you can drag WPS data file objects from any folder and drop them on the open editor - the result is that the file is added to the edit ring (the list of files currently being edited).
- If the program offers several tailoring options for the application, implement a *Settings* view, similar to those provided by WPS objects.
- If the application includes many large dialogs, currently using separate dialog boxes invoked from the menu bar, consider replacing these with a notebook control - it can make a complex application seem a lot simpler.
- Add an ASSOCTABLE statement to the program's resource script file. This statement sets up associations for the program so that the user is relieved of the need to do it himself, and also provides the only way to add new file types to those provided by the system. For details of this see 7.3.3.3, "Using an ASSOCTABLE to Add New File Types" on page 107.

  This statement allows for associations based on file extension, but our recommendation is to use file types as they provide greater flexibility for the user.

- If necessary, modify the program so that it behaves well when provided with an empty input file. For example, if the program normally expects the contents of its files to have some pre-defined structure, make the program set up this structure automatically if it reads an empty input file, rather than issue a message complaining that the file is invalid.

  The reason for this change is that we want the user to be able to start a new file by first dragging a new object of an appropriate type from the *Templates* folder, then opening it to start the application program. Typically, the new file will be empty at this stage, so the program must recognize and accept that the user has chosen to start a new one. Many existing PM programs expect the user to start a new file by selecting *New* from the *File* menu and will only load those files that already contain valid data.

For PM programs that operate on whole files, like word processors and spreadsheets, very good results can be achieved with these simple techniques, without ever having to write any SOM code. With a template file available, and associations already set up, all the user need do to start working on a new file (document, spreadsheet, or whatever it is), is to drag a new one from the template and open it. The program will start, presenting the user with a view of the data, and when he is finished working on it, he can close the view (that is, the program), and the data is saved automatically.

7.3.3.2 An Illustration

Let us illustrate this with a simple example. We have an existing PM program that is an expenses calculator. When it starts it presents an empty window with a menu bar that has two actions - *File* and *Process*. With the File action the user can elect to start a new expenses form, load an existing one from disk, save the current one to disk, or to exit. The Process pull-down menu includes actions to calculate totals, to print the current form, or to send it to the cashier for processing.

Let us see how we might modify this application to work in a way that is consistent with the Workplace Shell user interface:

1. Add an ASSOCTABLE statement to the program's resource script file, associating the program with the new type: *Expense Form*.
2. Modify the program so that it expects a filename as its first command line parameter and automatically loads an expense form from the named file. If the program finds that the input file is empty it should start a new expense form.
3. Modify the program so that when closed it automatically saves the current data into the file it read in when started (after prompting the user to confirm that this is what he wants).
4. Take the actions currently on the menu bar and implement them in a context menu displayed when the user presses Mouse Button 2. Provide an option for the user to turn off the menu bar, which is now redundant.
5. Provide the user with installation instructions that tell him to copy the .EXE file onto his disk and create a Program object referring to it. This will ensure that the *Expense form* file type is added to his system, and that an Expense form template file is created.

Now the user can start a new expense form by dragging one from the template in the *Templates* folder and double-clicking on it to start working with it. The

context menu will then let the user calculate totals, print the form or send the form off for processing.

This is still not perfect; the user may well expect, as with real Workplace Shell objects, to be able to access the context menu from the object itself, not just from a view of it. This can be put right by making two copies of the program, and modifying them as follows:

- The first should be modified so that it only prints forms
- The second should be modified so that it only sends forms for processing.

These programs will be easy to produce since they already contain all the required function. All that needs to be changed is to make them perform their specific functions automatically when started, and to remove extraneous code.

Since a large amount of code may now be duplicated between the three programs, it may be worth restructuring them so that common functions reside in a DLL accessible to all three. This provides significant benefits for run-time performance and program maintenance.

The user working with an expense form file will now find its context menu contains actions to open a form (which will start the main program), to print a form (which will invoke the new print program), and to send the form off for processing (which will invoke the other new program). These actions are also available from within the main program when the user is actually working on the form.

The user may find it odd that all these actions appear under the *Open* action in the context menu. This could be improved by removing the association from the Print and Send programs, and manually adding menu items to the template expense form file to invoke these programs.

The user will also find that he cannot print by direct manipulation of the file itself. If the user drags the Expense Form object onto a printer, it will print as a text file. There is no way that our print program can be invoked automatically to format the data in this situation. The only solution to this would be to develop a new WPS class, derived from *WPDataFile*, that overrides the *wpPrintObject* message. Then, when the object is asked to print itself, (typically in response to having been dropped on a printer object), it will invoke the print program to provide the required formatting. Developing this class would not involve very much work, but would require the programmer to be familiar with WPS and SOM programming.

With only minor exceptions the expense form file object now behaves almost exactly as it might if it had been implemented as a new Workplace Shell class derived from WPDataFile, when in fact it is simply an ordinary data file with some associated PM programs.

7.3.3.3 Using an ASSOCTABLE to Add New File Types

The scenario described above requires that a new file type be added to every user's system - in the example its name was *Expense form*. This section explains how to do this for your own program. An alternative approach that is suitable for users and administrators can be found in 6.8.5, "Adding New File Types" on page 84.

The standard programming technique to do this involves the ASSOCTABLE statement in a program's resource script file. (This is one of the source files that a programmer creates when developing a PM program - for more information about this see *OS/2 Version 2.0 - Volume 4: Application Development*). This statement defines associations for the program being developed, in terms of either file type, filename or extension. The file type can be anything you choose, regardless of whether it is one of the existing file types defined within OS2.INI. In the case of the example above, the *Expense form* file type may be used in the ASSOCTABLE statement, despite the fact that at that time it is not a type known to OS/2.

An ASSOCTABLE suitable for this example is shown in Figure 40. The first quoted string gives a file type to be used for association with this program; the second string allows for association by filename or extension (if you want to use only association by file type, an empty string may be specified here); the third parameter specifies that we want this program to be automatically associated with data files with this type; the last parameter specifies an icon to be used for representing data files that have this attribute.

```
ASSOCTABLE 5 PRELOAD
BEGIN
    "Expense form", "*.exp", EAF_DEFAULTOWNER, expenses.ico
END
```

Figure 40. An ASSOCTABLE Resource Script File Statement

When the program has been built, the *Settings* view for the executable file shows, on its Associations page, that associations have been set up for the specified file type(s) - in our case *Expense form*.

At this stage, however, there is no way you can create a data file with this file type - when you open the Type page of a data file's *Settings* view you find that *Expense form* is not one of the available types.

To achieve this you must create a Program object that refers to the executable file that you just created. (The usual way to do this is to drag a Program object from the Program template in the *Templates* folder, then insert the executable file's path and filename into the "Program" page of its *Settings* view). This registers the program to the Workplace Shell, which finds that the program includes a type for association, and adds this to its list of available file types. It also adds a template file for this type to the *Templates* folder.

The new types are then available to be added to any data files on the system and new files of these types can be created by the user at any time simply by dragging from the corresponding templates.

7.3.3.4 Converting a PM Program to WPS/SOM

To convert an existing PM application to be a full SOM/WPS application may require a great deal of work, depending on how well structured the program is to begin with.

Much of the code can probably be preserved - when a WPS object wants to display a window, it does so using normal PM facilities to create the window and has to provide a normal PM window procedure to support it; most of this code may be transferred directly from the older program. Similarly, any application

processing code - for example for database access, communications, or calculation should be equally applicable to the new version.

The steps necessary to make the conversion include:

- Consider the split between function that will be implemented as WPS objects (with all the drawbacks of running in the WPS process), and function that will be implemented in separate processes.
- Devise the interprocess communications (IPC) protocols to be used between the two parts of the application.
- Design the revised user interface. Although much may be preserved, there will inevitably be design changes needed to make the application work well for the user in the new environment. These may include:
  - Increased emphasis on the use of icons to represent the data items manipulated by the application
  - Changes to terminology used (for example, "views" rather than "windows")
  - Use of container windows where list boxes may previously have been used
  - Replacing complex dialogs with notebook controls
  - Adding context menus.
- Add the capability for the program to save its state when closed so that, the next time it is started, the program's options, settings, its window position and so on are the same as the last time it was run.
- Provide a way for the user to retrieve application windows that have been minimized. Since in OS/2 Version 2.0 windows are not necessarily minimized onto the desktop, it is possible that application windows that the program has not added to the window list may be hard to restore. Such designs need to be revised for OS/2 Version 2.0.

7.4 Summary

Presentation Manager applications differ in structure from conventional applications. Presentation Manager applications are composed of a number of windows, which represent objects upon which the application operates. Windows respond to events, communicated by way of messages passed to the windows from Presentation Manager or from other windows.

Windows may be either display windows, which operate on objects known as presentation spaces, or object windows, which operate on other objects such as databases or remote systems. Display windows have a visual manifestation on the screen, while object windows are typically hidden and act merely as addresses to which messages may be sent in order to initiate actions.

Windows are grouped into classes. Each class has similar characteristics and responds to messages in a similar way. Window classes must be registered to Presentation Manager. Some window classes are defined as public, and may be used by all applications in the system, while other window classes are defined privately by an application, and are available only to that application.

The response of a window to the messages it receives is determined by its window procedure. All windows of the same class share the same window procedure.

Messages are also grouped into classes; Presentation Manager defines a number of message classes, and applications may define their own classes for communicating events between windows in the application. Message classes are simply defined using integer constants, and are not required to be registered to Presentation Manager.

The structure of a Presentation Manager application is therefore that of a number of windows which communicate by way of messages. Since these messages normally constitute the only means of communication with a window and the objects upon which it acts, this application structure forms a good basis for the implementation of object-oriented software engineering principles. The use of such principles in Presentation Manager applications is discussed at length in *OS/2 Version 2.0 - Volume 4: Application Development*.

Workplace Shell applications are implemented using the System Object Model component of OS/2 V2.0. This is a language-independent environment for object-oriented programming, along with a set of utilities that enable the programmer to define object classes, and to implement them in non-OOP languages such as C.

A Workplace Shell application will typically consist of one or more objects, which are defined by the programmer using the SOM, and inheriting characteristics from the Workplace Shell's own existing classes.

The shell and all its objects run in OS/2 as a single process, which makes the shell and its objects vulnerable to a single object that is unresponsive or which abends. A well-designed Workplace Shell application should therefore be structured so that the bulk of its application logic runs in a separate process, communicating with the Workplace Shell objects that implements its user interface by means of OS/2 interprocess communication.

It is possible to write PM programs that integrate to some extent with the Workplace Shell environment, for example by supporting printing by direct manipulation, but such programs will never behave quite like properly designed Workplace Shell applications. Migrating existing PM applications to work better in the Workplace Shell environment is fairly straightforward, and good results can be obtained in many cases by exploiting the association facility of WPS. Fully migrating a PM program to become a SOM/WPS application may be possible in many cases, though this involves considerable redesign and programming.

We cannot give a clear recommendation as to which programming model - PM or WPS - to adopt for new applications. It will depend on many factors, such as the application itself, the needs and skills of the users, the skills of the programmers, and on any other applications that may be used at the same time. The decision must be made based on the nature of the application and the customer environment in which it is to be used.

Chapter 8. Workplace Shell Implementation

This chapter looks at what happens "under the covers" of the Workplace Shell (WPS). We examine the shell from several viewpoints:

- The implementation of the WPS as an OS/2 program
- The different types of shell objects
- The relationship of the shell to the file system
- Persistence in WPS objects.

This chapter will help you to understand how the Workplace Shell is implemented, know what kinds of objects it stores and where it stores information about them. If you wish to understand the System Object Model, object-oriented programming in OS/2 V2.0 and how to create your own WPS program you should read the appropriate sections of the companion volume, *OS/2 Version 2.0 - Volume 4: Application Development*.

8.1 The Workplace Shell as an OS/2 Program

One of the biggest changes in OS/2 V2.0 is its support for installable interface shells. The installable feature concept was introduced in OS/2 Version 1.3 with the HPFS file system; OS/2 V2.0 has now extended it to the user interface. The installable feature approach gives users the flexibility to set up their system to meet their precise needs.

The Workplace Shell is called from the CONFIG.SYS file as follows:

```
PROTSHELL=C:\OS2\PMSHELL.EXE
```

Alternative shells can be implemented or, if desired, a single program, such as Lotus 1-2-3 /G, could be started directly without having to start the WPS at all.

The Workplace Shell run in its own process, starting threads as needed. For instance, the contents of a directory can be read while its associated folder window is being created. In addition, the WPS starts a second process to monitor the shell and restart it if it fails.

This in turn means that the overall system is no longer as vulnerable to shell errors as was the case in all previous versions of OS/2. If it detects an error, the WPS is capable of restarting itself without having to shut down and restart all the other running programs.

The WPS does this by saving all the window information in real time and restoring it when the shell process is restarted.

Existing PM applications are unaffected. PM programs which are implemented with a Model/View structure, where the "view" part runs in the WPS process, will have to be restarted and reconnected to the "model" portion of the program running in its own process.

The Workplace Shell also notes which programs are running at any time and can restart them when the operating system is re-IPLed.

8.1.1 Workplace Shell and the System Object Model

The other major new feature of the Workplace Shell is that it is built on an object-oriented platform, called the System Object Model (SOM). SOM enables the WPS to be written as classes which inherit attributes from their superclasses and provides a high degree of reusability and integrity to the WPS implementation. Object-oriented programming and SOM are described more fully in Chapter 7, "Presentation Manager and Workplace Shell Application Development."

8.1.2 Workplace Shell Object Types

There are three main classes in the WPS class hierarchy, descended from *WPObject*. The WPS classes are shown below:

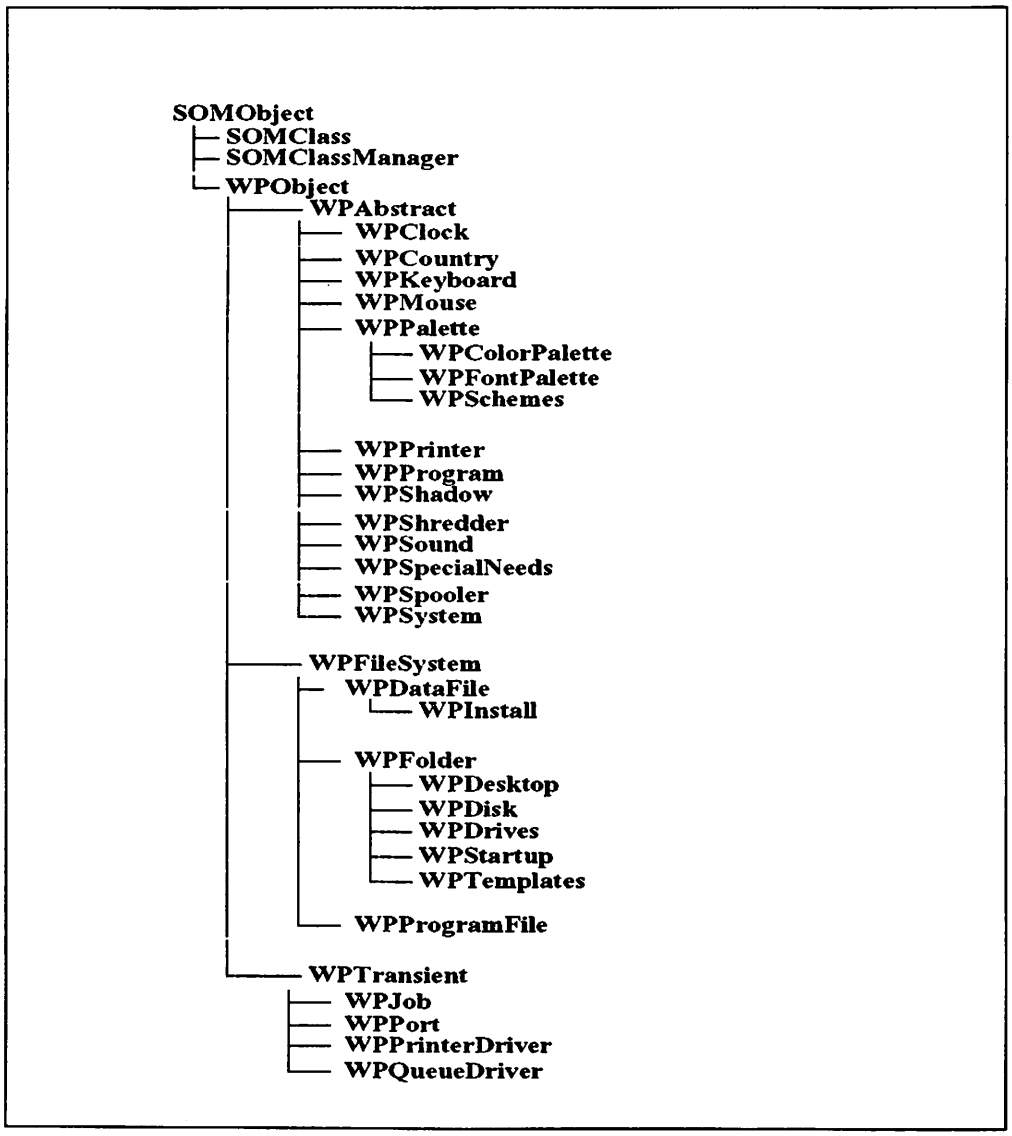

Figure 41. Workplace Shell Class Hierarchy

Classes are used to create the **instances**, or objects, that the user sees on the desktop. Each can be **subclassed** to create a new, derived class which inherits some of the properties and behaviors of the parent. Two useful terms to note here are **base class**, a class which is a direct subclass of *WPObject*, and **persistence**, which denotes that an object knows how to save its current state.

The main WPS classes are:

WPObject The **superclass** from which all base classes are derived.

WPAbstract A base class that provides persistence via the OS2.INI file. These include programs, shadow copies and system objects such as the clock. They can be copied around the Workplace Shell but not to diskette.

WPFileSystem A base class that provides persistence via the .CLASSINFO extended attribute of the associated file or directory. Objects of this class are always stored on disk. They are typically folders and files and can be copied to any media.

WPTransient A base class that has no persistence. Examples of *WPTransient* objects are the window list and the printer drivers.

8.1.3 Relationship of the Shell to the File System

The WPS represents program and data files with icons and allows you to move and copy them around the system in a similar way to the OS/2 Version 1.3 File Manager. There are also new functions like "shadow copy" and new objects like the system clock which have behaviors completely unlike those previously seen in the File Manager.

The implementation of the Workplace Shell in OS/2 Version 2.0 is not inherently file-oriented but the current implementation only supports files, not data records or transactions. Data file and program icons may represent files on a disk. A folder is represented by a real directory under its name. The diagram below shows the approximate disk structure which supports the WPS:

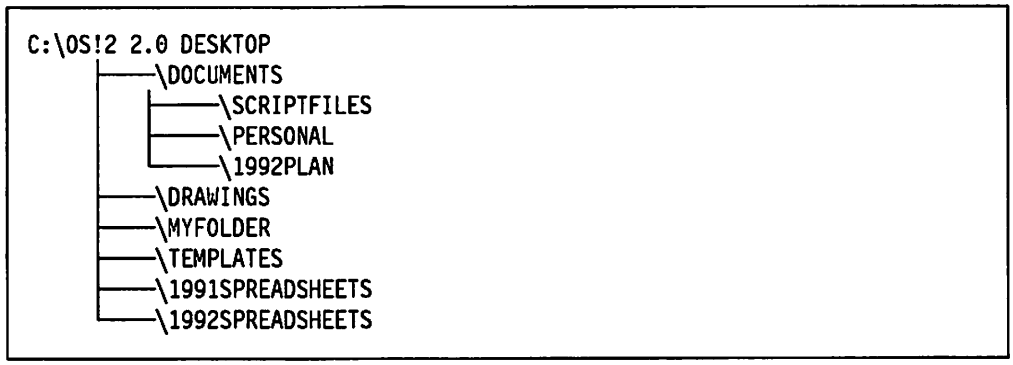

Figure 42. Disk Structure Supporting the Workplace Shell

As you can see, all folders are contained within the desktop and the directories of each folder on the desktop are subdirectories of the "OS!2 2.0 DESKTOP" directory. From the directory structure we can see that the screen will display a *Documents* folder on the desktop and it, in turn, will contain three other folders; SCRIPTFILES, PERSONAL and 1992PLAN.

Icons in any folder can be dragged (moved) into another folder or onto the desktop. When this happens, the file system will handle it in two different ways. If it is a data file it will be physically moved to the new directory. If it is a program reference or a shadow copy of a data file, then a pointer to the original object and its working directory is passed from one folder to the other.

The relationship of the desktop to the file system is shown in Figure 43 on page 114 below.

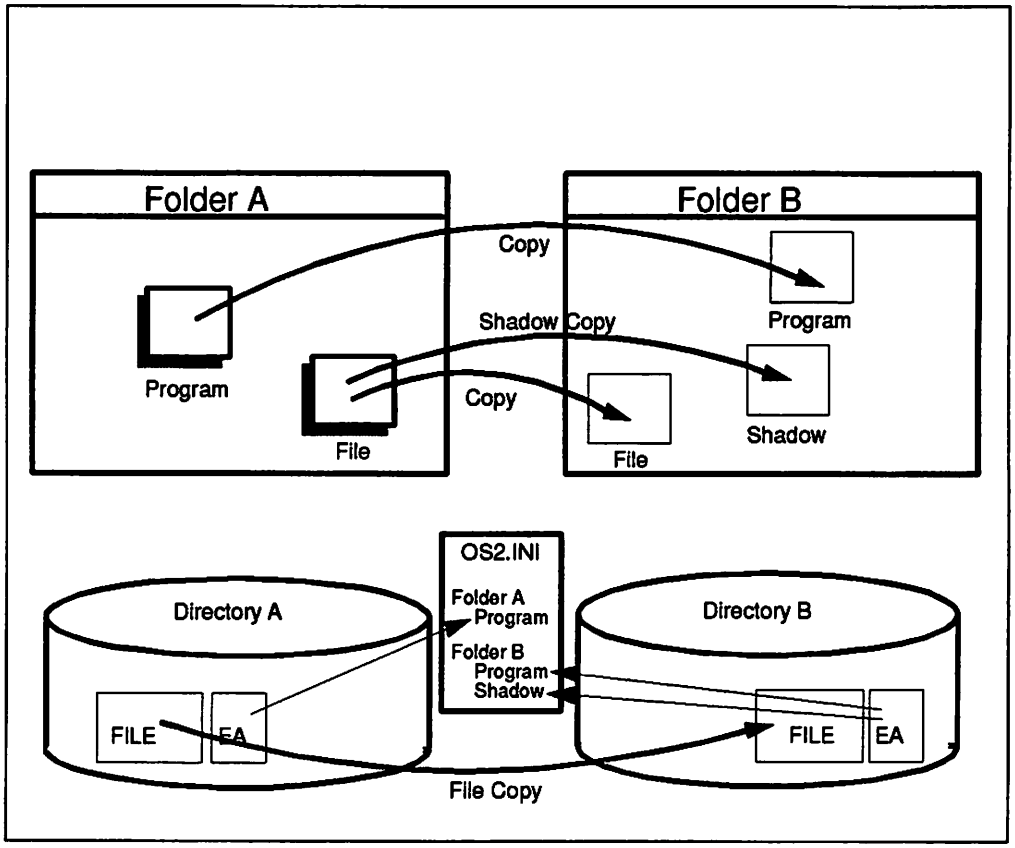

Figure 43. Drag/Drop - Physical Implementation

Both the folder and the abstract objects in it are stored as pointers (called *HOBJECT*s) in the OS2.INI file (see 8.5.1.3, "Folder Contents" on page 127 for more details). When a program is copied to another folder, the HOBJECT of the program is placed in that folders area within the OS2.INI file. The HOBJECT is also placed in the EAs of the directory which corresponds to the folder, along with its position within the folder.

Handling programs in this manner makes sense because, if the programs' main executable modules were to be moved, the executable files could become separated from their DLLs. This approach allows a program to be copied and moved around the desktop without having to be physically moved.

"Shadows" of file objects are handled in a similar way to programs; the original file is left in the source folder but a pointer to it is created in the target folder. HOBJECTs are discussed further in 8.2.1, "Folders."

There are some problems associated with the Workplace Shell implementation. These are outlined below and discussed in more detail in the rest of the chapter:

- The system is critically dependent on a single file, OS2.INI, for much of the information. Damage to this file can have a disastrous effect on the user's working environment.
- The WPS uses EAs to store settings for files. However, EAs are not recognized by all file systems, nor by any DOS or Windows programs. This can cause problems when setting up the working environment for a user who needs both OS/2 and DOS programs.

- With a FAT file system, moving files can lengthen disk access times as files become fragmented and the tables become cluttered. This would impact users who want to migrate their systems from DOS to OS/2 V2.0.
- A program file is treated by the WPS as either a program reference object or a data file object, depending on the view used. Once registered as a program reference object and placed in a folder, the WPS restricts the actions which can be performed on it. For example, copying or moving the object moves the program reference, not the underlying file. Viewing the object from the *Drives* folder, however, allows the user to work with the physical file. Thus direct manipulation in the WPS means that sometimes the program file can be moved and sometimes it cannot.
- While data files and programs are handled by the shell, there is no base class for record structures and transactions. To be able to create a persistent "transaction" object capable of interacting fully with the other WPS objects might require us to derive a new base class from *WPObject*. See *OS/2 Version 2.0 - Volume 4: Application Development* for a more detailed discussion of this issue.

8.2 Workplace Shell Objects

This section looks at the main Workplace Shell objects and provides an introduction to how they are implemented.

8.2.1 Folders

Every folder corresponds to a directory in the file system. The desktop is a special type of work area and is represented by \OS!2 2.0 DESKTOP in the file system. The top-level folders on the desktop are subdirectories of \OS!2 2.0 DESKTOP. *WPFolder* objects and their contents are stored using a combination of a directory and the OS2.INI file. A folder may contain any kind of WPS object.

WPFileSystem objects are real files residing in a file system directory corresponding to the folder. These objects are stored in the folders directory but not in the folders section of the OS2.INI file.

WPAbstract objects are stored in the OS2.INI file. The information is stored with a pointer, called a *HOBJECT*. The folder stores the HOBJECTs for the abstract objects contained within it in the OS2.INI file, as can be seen below in Figure 44 on page 116. This information is also stored in the EAs file for the directory associated with that folder. See 8.5, "The OS2.INI File" on page 126 for more details and some examples.

WPTransient objects can be placed in a folder but are not stored by the operating system and disappear from the folder when OS/2 is shut down.

Settings information is stored in the Extended Attributes of the folder's directory. The following figure may help to illustrate these relationships:

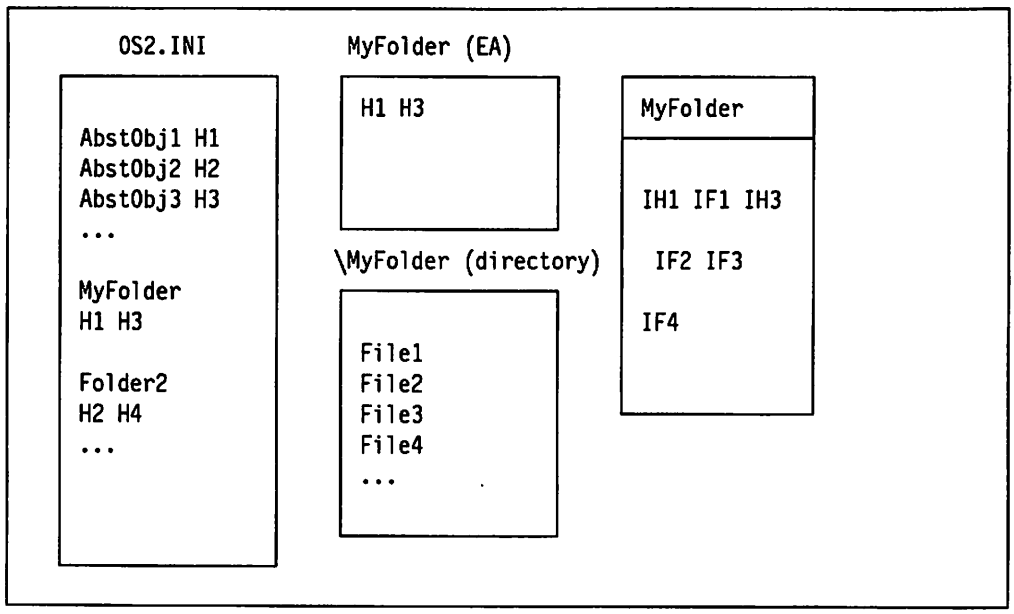

Figure 44. Relationship of Folder to Directory and OS2.INI File

In the folder, *IH1* is the icon for a program, *IH2* is the icon for a shadow copy of a file, and *IF1* to *IF4H1* are icons for data files. The data files are stored in a directory - in HPFS this bears the same name as the folder. Pointers (HOBJECTs) to the abstract objects are stored in the Extended Attributes (EA) for the directory.

The pointers are generated by the WPS when an abstract object is created (for example, by installing a program) and are unique to each *WPAbstract* object. The physical references to each *WPAbstract* object, together with its HOBJECT, are stored in the OS2.INI file. We can see that the program represented in the folder as *IH1* is called *AbstObj1* in the OS2.INI file and has a HOBJECT of H1. This HOBJECT is what is stored in the directory EA. You will notice that OS2.INI also separately records which *WPAbstract* objects are stored in each folder.

8.2.1.1 Folder Population

A folder populates itself with icons when it is opened and refreshed. The following approach is used:

- The *WPAbstract* class keeps track of which folders its instances are in.
- Pointers to *WPAbstract* objects (programs, shadows, etc.) are read from the OS2.INI file and their icons are displayed.
- The *WPFindObjects* method of *WPAbstract*, *WPTransient*, *WPFileSystem*, and any other base class is used to retrieve the contents of a folder. For example, in *WPFileSystem* this method reads the EAs for each file in the directory, which tells it the object's class and provides its icon. A message is sent to the appropriate class which then instantiates it. If the file doesn't have any EAs then the default used is the base class, *WPFileSystem*.
- The EAs are read for the directory. This gives the name (or OS2.INI program reference) and icon position for each object to be displayed in the folder.
- The object icon and title are displayed in the folder.

Any alteration to the contents of a folder, such as adding a file or removing a shadow copy, is saved by the object when the event occurs. However, attributes of the folder, such as icon positions or the size and position of an opened view of the folder, are not saved until the contents view of that folder is closed.

A folders view will not be automatically updated when a file is created or destroyed by a process outside the WPS process (such as from a command line in an OS/2 window). Restarting the system may resolve this.

This is because the WPS does not receive every message from the file system concerning files which lay outside its workplace directory structure (that is, not under "OS!2 2.0 Desktop"). Notification is received if a new file is created or if an existing file is deleted or renamed, but not if an existing file is changed outside of the workplace.

8.2.2 File System Objects

A data file object in the Workplace Shell represents a real file in the file system. If the user shreds the object by dragging its icon to the shredder, the file is deleted.

The only view which is always available to a data file is the *Settings* view. Settings are stored in extended attribute (EA) files. Refer to 8.4, "Extended Attributes" on page 124 for more information on EAs. Settings are displayed in a separate window from the main window, using a notebook control.

The difference between descriptive and physical names is important. The *descriptive* name (or "object title") is the name which appears under the icon. It can be set in the "Title" field in the *Settings* view, or by "direct editing." The *physical* name is the actual name by which the data file is known to the file system. It can be set in the "Physical name" field in the *Settings* view.

The two need not be the same. This provides an advantage in that long, meaningful icon descriptions can be used even without HPFS.

The filename and object title are synchronized as much as possible. However, in FAT file systems the physical name is limited to 12 characters. When the description is longer than FAT will support, the filename is truncated. If a similar filename already exists, a version number is appended to it, However, the WPS always tries to ensure that the digits at the end of the filename match those in the title of the object. For example, a file with the title "My File : 22" might have a filename of "My_Fil22" on FAT and "My File : 22" on HPFS.

There is some possibility for confusion in both file systems, since two objects within the same folder can have the same descriptive name even though they have different physical names. Figure 45 on page 118 shows the effect of "direct editing" an icon description.

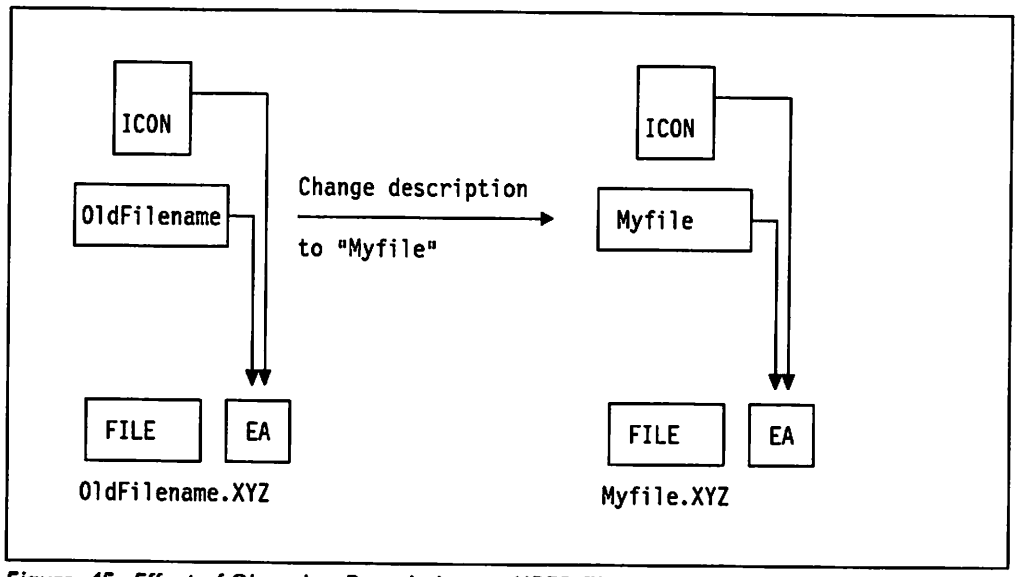

Figure 45. Effect of Changing Description on HPFS file names

This is another reason for using the HPFS disk format: if you change the file description the WPS automatically changes the filename to be the same as the description. As you can see, the description and icon for the object are stored in the files Extended Attributes (EAs). When you copy the file under OS/2 V2.0, its EAs are copied too.

The operating system settings determine what happens when you try to copy a file into a directory which already contains a file of that name. If a user copies an object to a folder where an identically named object already exists, the Workplace Shell will prompt the user to change the name (this is the default setting). The WPS applies a similar protection if the user tries to rename the object using "direct editing" of the icon description, but will not prevent the user from changing the title in the *Settings* view.

For example, if a user copies a file to another folder, changes the contents, then copies it back to the original folder, the filename will be changed as follows:

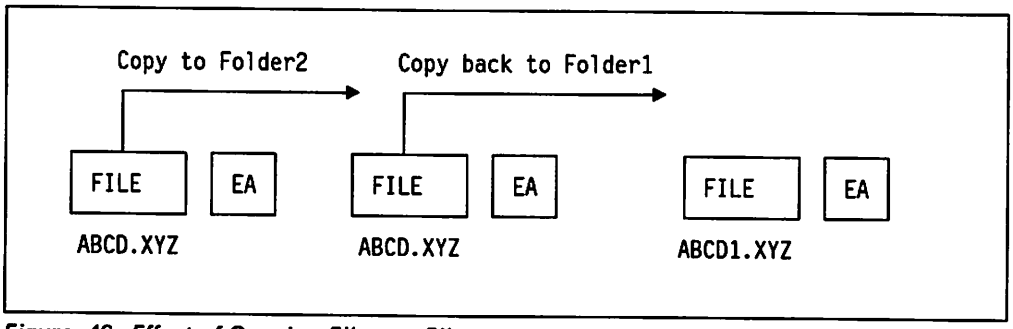

Figure 46. Effect of Copying Files on Filenames

When you copy the file from FOLDER1 to FOLDER2, the filename stays the same (ABCD.XYZ). When you copy it back to FOLDER1, the user is prompted to change the filename to ABCD1.XYZ. This is because the WPS believes that it should not destroy user data by default.

Sometimes this is very useful. For instance, if you copied a spreadsheet, changed it extensively but forgot to rename it, then copied it back, you would be

horrified if you then realized you had just copied over the original - which was just as important as the new one!

A data file may associated with an executable file. This allows the program to be started and the data file passed to it when the data file is double clicked on. This is discussed in more detail in 8.3.2, "File Associations" on page 121.

Making a filename association in a program and setting the file type attribute in an associated data file can sometimes cause problems. This is because filename associations take precedence over file type associations. So if a user subsequently wants to change the file type for an object to use a different program with it, the file extension association will prevent him from doing so.

8.2.3 Shadows

Shadows are "aliases" for objects. Both *WPProgram* and *WPFileSystem* objects can be "shadowed" but it is not recommended to make shadow copies of programs. This is because a user might mistakenly edit the program title of the shadow copy and change the original program's filename, thus preventing it from being executed.

The behavior of a shadow is identical to the object it shadows in all respects except one; deleting a shadow does not affect the source object. In practice, the user may find other differences; for example, deleting an object causes the deletion of all its shadows. Shredding an object which has shadows linked to it will generate a message window, informing the user that this object has shadows associated with it.

Distinguishing a shadow icon from an object icon can be difficult, as there is only a subtle visual distinction between objects and their shadows (the text of a shadow is "grayed" or "dimmed"). Some users find it difficult to distinguish such a subtle difference and they should modify the shadow color text using the *Scheme Palette* within the *System Setup* folder. Shadows are also differentiated from real objects by the presence of the "Original" choice at the bottom of their context menus.

8.2.4 Abstract Objects

These include program references and shadow copies of files. Some can be programs written and executing within the Workplace Shell, others can be application programs running in their own processes while others can be shadow copies of data files. Since all objects have to have a program running behind them somewhere, there are many of these objects in the system, stored in the OS2.INI file.

The system clock is an example of a program running in the WPS process, while the icon editor is an example of a PM program running in its own process outside the WPS. Shadow copies are discussed in 8.2.3, "Shadows."

Since *WPAbstract* objects only physically exist in the OS2.INI file, they cannot be copied to diskette. This may confuse some users, who think that dragging a program reference or shadow copy to a diskette will cause the real file to be copied.

Information about *WPAbstract* objects is held in the OS2.INI file. For more information refer to 8.5, "The OS2.INI File" on page 126.

8.2.5 Transient Objects

The main difference between these programs and abstract objects is that transient objects cannot save their state, that is, they are not "persistent." Also pointers to transient objects are not stored in directory EAs, unlike abstract objects. The values used when transient objects are started are usually stored within the program and there is no mechanism to write these back to disk. Examples of these programs include the window/task list and device drivers.

8.3 Workplace Shell Facilities

In looking at how the Workplace Shell is implemented, it is useful to separate the shell from the underlying workplace facilities, since the same facilities are needed by both the shell and by all applications which run under the shell. These facilities include:

- Object registration
- File associations
- Direct manipulation
- Icon interaction/messaging
- The ability to create multiple instances of objects.

These functions are implemented as WIN APIs which can be accessed from any language that has appropriate bindings. For example, "WinCreateObject" can be called from a C program. Some of these APIs, such as "SysRegisterObjectClass," have been mapped to a new version of REXX,

One of the most important points to make here is that, from an application's viewpoint, the WPS can be totally invisible provided the application only uses objects which are already defined in the WPS class hierarchy, such as data files. For example, templates for data files can be created and associated with the program.

Applications written for previous versions of OS/2 may take advantage of the basic functions supported by the class under which they are registered (typically *WPProgram*). For instance, a program can be started by dropping a data file on the program icon providing the program will accept a file name as a parameter.

8.3.1 Object Registration

When the user double clicks the mouse on a data file icon in the Workplace Shell, a window opens up to display its default view. To do this, the system has to invoke a program and pass the file name to that program as a parameter. This requires the system to know which application (or object handler) to invoke, and where it is stored.

In the File Manager under OS/2 Version 1.3, the services which performed this function were hidden and not available to applications. In OS/2 Version 2.0, however, they are exposed and accessible through the Workplace Shell. There are three elements to this: program registration, file registration and file association.

When an application is installed under the Workplace Shell, a program template is used to register the application as an instance of the *WPProgram* class (a subclass of *WPAbstract*). This makes the program a persistent object.

The program reference information and icon are stored in the OS2.INI file. The icon is displayed in a specified folder, and a default view is defined. After registration this icon can be dragged and dropped on other WPS objects, such as a folder or the shredder.

Data files are also registered with the WPS, which creates new instances of the *WPDataFile* class for them. The file is stored in a directory and its "settings" information is stored in EAs attached to the file.

When you make a shadow copy of a file, the WPS creates an instance of the *WPShadow* class and stores it in the OS2.INI file. For this shadow, as with programs and files, using WPS facilities to create a new instance automatically registers it with the WPS.

8.3.2 File Associations

Files are associated with their "file handler" in two ways. The *Settings* view for an executable file contains an *Association* page. This page allows an application to be associated with its data file. The association can be by file type or by full or partial file name and extension.

The file type, file name, partial file name or file extension (for example WP*, WP*.DOC, *.DOC) can be associated with a program from the program itself or the program reference. The file type (for example, OS/2 command file or plain text file) can be associated with a program using the *Settings* view for an object. Both sets of information are stored in the OS2.INI file. See 8.5, "The OS2.INI File" on page 126 below for further details.

A data file that "belongs" to an application will reflect this by having the application's name appear as a view that can be opened. Plain text files, for example, may be opened as a "System Editor" view. The *Association* page is available for command (CMD) files as well as executable (EXE) files.

This duplication can cause confusion to the user if the associations are set in all three places. We recommend that if you have a program reference for a program, use this to set the associations instead of setting the associations from the program object itself.

8.3.3 Direct Manipulation

In OS/2 Version 1.3, a set of functions and messages were provided for direct manipulation, but each program had to provide its own code to take advantage of these functions, and define its own protocols to be observed during drag/drop operations, so that each program knew what type of information was being transferred.

These protocols are called *Rendering Mechanisms*. Three standard ones were defined in OS/2 Version 1.3:

Print Provided a mechanism to enable printing by direct manipulation

OS/2 File Intended to be used by PM programs that wanted to move and copy files by direct manipulation (such as the File Manager)

DDE Enabled Dynamic Data Exchange links to be established by direct manipulation.

Since PM did not provide common objects, such as printers and shredders, to implement these protocols the value of these rendering mechanisms was

reduced. The apparent complexity of the direct manipulation functions and protocols also inhibited their use.

More importantly, however, the non-object-oriented structure of most PM applications made designing direct manipulation into existing programs both difficult and largely ineffectual. The benefit of direct manipulation is directly proportional to the granularity of the objects being manipulated; where the finest grained object is a file, the only other objects it can interact with are containers and devices. Such interactions can as easily be implemented by the shell, as is shown by the implementation of the WPS.

OS/2 Version 2.0 provides many common objects which can be used by applications, all of which are drag/drop enabled. When an icon (representing a WPS object or something dragged from a PM program) is dragged over a WPS object, that target object will send a message to the source object to try to complete a direct manipulation operation.

That message says what the target icon does, so a successful drop depends on the source and target object implementing a common rendering mechanism; that is, the source object must be able to:

- Understand the message
- Perform the action carried in the message.

If the source icon represents something that is unsuitable for dropping on this target object, then a drop will not be allowed and the user will be informed by the pointer changing to a "no-entry" symbol.

Where possible, the WPS includes standard PM rendering mechanisms (for example, DM_DISCARDOBJECT) as well as its own WP messages (for example, (WP_DELETE). It is therefore possible for a PM application to interact with WPS devices by coding appropriate responses into the application using the drag/drop APIs. To do this, the programmer must know and understand the rendering mechanisms used by each WPS object.

This is made more difficult because the WPS, for performance reasons, often implements functions within the core classes rather than leaving it to objects derived from the base classes, such as *WPFileSystem*, to carry out the action. Thus, when dragging a file to the shredder, the file does not receive the WP_Delete message; instead, the shredder program tells the OS/2 file system to delete the file.

8.3.4 WPS Events/Messages

Workplace Shell makes implementing drag/drop considerably easier for the programmer than it is under PM. Much of the filling of data structures and the drag part of drag/drop is handled by WPS, so that WPS simply sends the target object a pre-defined message if a successful drop sequence is completed.

Without this help from WPS, a program must, for example, include code to handle the DM_DRAGOVER messages that occurs when an icon is being dragged over another icon without being dropped on it. If each application had to code an exchange for each device object in the system, as well as all the other possible objects, this would significantly complicate application development. It would also increase program size and memory usage, since each application would need to be running all the time.

The Workplace Shell avoids this requirement by allowing a developer to register a small object which understands all the capabilities of the application, and can call the relevant parts of the main application when required. The standard direct manipulation capabilities of the WPS include Print, Delete and File Move/Copy.

8.3.4.1 Print

Workplace Shell implements the standard PM Print rendering mechanism (DRM_PRINT); after a successful drop, the source icon is sent a "DM_PRINTOBJECT" message with the name of the relevant print queue as one of its parameters. For WPS objects, the WPS may print the object automatically; non-WPS programs should perform the print to the specified queue when they receive the DM_PRINTOBJECT message.

8.3.4.2 Delete

The Workplace Shell shredder object supports a rendering mechanism called DRM_DISCARD. This is not one of the standard PM rendering mechanisms. If an item supporting this rendering mechanism is dropped on a shredder, then the source program is sent a "DM_DISCARDOBJECT" message. In the case of a Workplace Shell object, it will be deleted; in the case of a non-WPS PM program, the program must delete the dragged item in response to this message.

8.3.4.3 File Move/Copy

When an item is dropped onto a folder object and the dropped item is either a WPS object derived from *WPFileSystem* or a PM item whose associated data structures indicate that it is suitable, the target initiates the OS/2 File rendering mechanism (DRM_OS2FILE). In the case of the PM item, its suitability depends on whether its associated DRAGITEM structure includes a reference to DRM_OS2FILE in its hstrRMF field. For details of this see *OS/2 Version 2.0 - Volume 4: Application Development*, which includes a detailed discussion of direct manipulation.

In both cases, copy and move operations are supported according to any modified keys the user may be holding down at the time. The implementation of this rendering mechanism results in files being moved or copied between directories on disk.

8.3.5 Persistence

Some objects will disappear when the machine is rebooted, others will reappear. The former are called "transient" objects in the WPS and the latter are called "persistent" objects. Persistent objects are stored in the OS2.INI file and directory EAs.

When you close a work area or shut down the desktop, any opened objects are recorded in the OS2.INI file and restarted next time the desktop or work area is reopened. Persistence for all WPS objects, such as the system clock, programs and shadow copies, is handled by their classes.

The OS2.INI file implementation is critical to the WPS because it represents a single point of focus for the entire operating system. If it becomes corrupted the system will lose all the information about how to run programs, the associations between files and programs and which programs were previously running in the system.

8.4 Extended Attributes

Extended Attributes EAs) are widely used in the Workplace Shell to record information about the attributes of the WPS objects. In general, information about an object is stored by the object itself. Thus a file stores information about its class and the location of the icon used to represent it, while the folder stores the position of the icon within it.

On an HPFS partition, Extended Attributes are stored in a special, hidden area, close to the files themselves. On a FAT file system, Extended Attributes for files and directories are stored in a hidden file in the root directory of each FAT partition.. This file is named "EA_DATA. SF."

EAs for the Workplace Shell are stored in the root directory of any logical disk accessed by it, in a hidden file called "WP_ROOT. SF." This file holds specific information about setup of the desktop. It is updated after the disk is accessed by any WPS object, such as the *Drives* folder.

EAs for the LAN independent shell are stored in the root directory of any logical disk accessed through LAN utilities, such as the *LAN Server* folder, in a hidden file called "WP_SHARE. SF." This file is updated when changes are made to the logical disk by programs which make up the LAN independent shell.

However, changes which are made to this disk from a WPS program are stored in the "WP_ROOT. SF" file. For example, this would happen if a user accessed the disk by issuing a "NET USE" command and then used the *Drives* folder to work with it.

8.4.1 Directory Extended Attributes

Folder attributes are stored in the Extended Attributes (EA) of the directory associated with it. The WPS creates two sections in the EAs for a directory. One section is used to store the icon positions for the objects displayed in the folder while the other records the folder attributes. This section discusses the directory EAs on an HPFS disk.

The first one created is called ICONPOS and contains the following information:

```
..A...WPShadow:A2DA0.a..
..J...WPDataFile:Fxyz...
..J...WPDataFile:Fabc...
..J...WPDataFile:FData_File.V.
..<...WPShadow:A48CA.@...C....
```

Figure 47. Contents of Directory Extended Attribute (ICONPOS)

This folder contains five objects: three are data files, called "xyz," "abc" and "Data_File"; and two are shadows, with HOBJECTs "2DA0" and "48CA."

The ICONPOS section is created automatically by the WPS and 30 bytes are allocated for the folder position. Each file which is added to the folder is recorded here and 21 bytes are allocated for the icon position plus 1 byte for each character in the file name. Abstract objects (shadows and programs) take a fixed number of bytes; 23. This is based on 21 bytes for the icon position and 2 bytes for the abstract reference (HOBJECT) in the OS2.INI file.

The other section is called "CLASSINFO." This includes information such as the class the folder belongs to (this can be changed by the "work area" checkbox in the *Settings* view).

```
.............WPFolder.E)...)..
.)... .....(.WPFolder...T.....
0...U...D.P...[.....0.........
....6.WPObject................
.............................
```

Figure 48. Contents of Directory Extended Attribute (CLASSINFO)

Information about which WPS class the object belongs to is needed to ensure that the correct class methods are used when the WPS is asked to create a new instance of the folder.

Directory EAs are updated when the folder settings are changed from the *Settings* view and as the name or contents change.

8.4.2 File Extended Attributes

File settings are stored in extended attribute (EA) files. EAs are automatically created if the file was created using one of the WPS templates, but not if the file was copied or created directly from a command prompt.

If the *Settings* view is opened and the EAs do not exist, then they are created at that time. When any of the settings are changed, the EAs are updated. This is done when a page on the notebook is changed and when the *Settings* view is closed The settings information in a file's EAs looks like this:

```
EA #: 1  NameLen: 10  Name: >.CLASSINFO< ValueLen: 142
.........w...WPDataFile..rJ..rJ.-sJ.g.....T.WPObject.............

EA #: 2  NameLen: 9  Name: >.LONGNAME< ValueLen: 15
....Test 1 File

EA #: 3  NameLen: 8  Name: >.SUBJECT< ValueLen: 27
....Test file for LAN stuff

EA #: 4  NameLen: 5  Name: >.TYPE< ValueLen: 34
..........Plain Text....BASIC Code

EA #: 5  NameLen: 5  Name: >.ICON< ValueLen: 4070
```

Figure 49. Contents of File Extended Attributes

The file EAs are used to store:

Classinfo This contains the WPS class to which the object belongs. This ensures that the correct WPS methods are used to instantiate the object when the folder is opened. Any new programs added to the file menu are also stored here.

Longname This contains the descriptive text that is displayed below the icon. This is needed for FAT file systems where filenames are restricted to 8 characters plus 3 for the file extension.

Subject This data is entered in the "Subject" field of the *Settings* view.

Type This records the file type of the object. A file can have multiple file types. They are used to provide an association with a program through the OS2.INI file.

Icon This stores the new icon used by the object if the user decides to replace the default icon. Note that if he chooses to "Create" a new icon, 1014 bytes are used, whereas if he "Edits" or "Finds" an icon, 4070 bytes are needed.

File EAs can shrink as well as grow. The disk space needed for the EAs grows as options are chosen but if the choices are removed later the EAs size will be reduced.

8.5 The OS2.INI File

This is the most important file in the Workplace Shell. It is the only place where information about *WPAbstract* objects, including which folder(s) they can be found in, is stored. The need for sensible backup and recovery mechanisms for OS2.INI is discussed in Chapter 6, "Installing and Supporting the Workplace Shell" on page 67.

Because it is the only place in OS/2 V2.0 where *WPAbstract* objects are stored, the Workplace Shell is limited to those *WPAbstract* objects defined within the OS2.INI in the root directory. That is, the Workplace Shell user cannot see any *WPAbstract* objects defined on a server, even if he can access the entire disk and desktop structure of that server.

The OS2.INI file uses its own data structure and cannot be edited by an ordinary text editor, so if anything goes wrong you will probably have to reinstall OS/2 Version 2.0. Taking a backup of the OS2.INI file is highly recommended. Running MAKEINI.EXE is a possibility, but this will only restore the OS2.INI file to the state it was in when it was originally installed and any subsequent customization will be lost. These approaches may result in HOBJECT pointers in the directory EAs being different from those in OS2.INI, leading to a condition known as "unresolved abstracts."

The OS2.INI file includes the following information:

- Running programs
- Shadow copy objects
- Program reference objects
- Icons for abstract objects
- Color palette objects
- Disk objects
- Object instances in each folder
- Folder position
- File/program association filters
- File/program association types

8.5.1.1 Running Programs

When the operating system starts up it restarts any programs which were running when the WPS stopped. The OS2.INI file contains the necessary information to support this. For example, Figure 50 shows several running programs:

```
FolderWorkareaRunningObjects
        D:\OS!2 2.0 DESKTOP\OS!2_SYSTEM\COMMAND_PROMPTS
                WPProgram:A74A6.
        D:\OS!2 2.0 DESKTOP\OS!2_SYSTEM
                WPFolder:DCommand_Prompts.
                WPDrives:DDRIVES.
```

Figure 50. Running Programs Stored in OS2.INI

The running programs can include programs and workplace objects, such as folders and drives.

The above figure shows two open folders, *OS!2_System* and its subdirectory *COMMAND_PROMPTS*. The running programs are stored together with their location so that it is easier to reinstantiate the system as it was left. Refer to the process of opening folders in 8.2.1, "Folders" on page 115 to understand why this is so.

8.5.1.2 Abstract Objects

Each abstract object is also stored in the OS2.INI file so it can be located using its reference. In the following example, we can see how a program is stored with its instance data:

```
PM_Abstract:Objects
        D98F
                ....WPShredder..#..p#...#......
                ....WPAbstract.......Shredder..
                ..........G.WPObject..........
                ..............................
                ..........<WP_SHRED>...........
```

Figure 51. Abstract Object Reference for Shredder in OS2.INI

This object is the WPS *Shredder*. It is stored with its Workplace Shell class, OBJECTID, icon description and the classes it inherits from. The hexadecimal pair, D98F, in the upper left corner is the HOBJECT of the shredder. The location of each abstract object is also stored within the folder section of OS2.INI, as can be seen in the example in 8.5.1.3, "Folder Contents."

8.5.1.3 Folder Contents

Folder contents are stored in two places; in the ICONPOS EA file for each directory and in the OS2.INI file as shown below. Any running programs in opened folders are stored under the *WorkAreaRunningPrograms* section for reinstantiation during the OS/2 initialization, while the *FldrContent* section below includes all the abstract object references in that folder. These references are also stored, with their window coordinates, in the ICONPOS.EA file of the directory associated with the folder.

```
PM_Abstract:FldrContent
       5506
              0000  8FD9 0000 AB91 00 00
              0008  1C04 0000 1E7E 00 00
              0010  E13E 0000 22D5 00 00
```

Figure 52. Abstract Objects Contained in Folder 5506

The hexadecimal numbers are read as pairs, for example 8FD9 is actually D98F. Each pair is a pointer to an abstract object (or program) in the system. Here, we can see that the shredder (D98F) is referenced as the first item in the folder. To help you read this, it is useful to know that

- 5506 is the HOBJECT for the folder
- D98F is the HOBJECT of a WPAbstract instance contained in this folder.

8.5.1.4 File Associations

There are two association mechanisms in the OS2.INI file; one for association filters and the other for association types. The format for association types works with the information stored in the settings EAs for each file. If a file doesn't have any EAs then the association is based on its full or partial file name or extension, using the association filter.

There are two formats for the association filters, depending on whether the program runs in the WPS process or its own process. These trigger the appropriate program when you double click on a data file icon. The first filter, for any file with a **.MET** extension (metafile), shows that two programs can be used. The first program is described by an OS2.INI program reference (8134). The second program includes the pathname and file name (PICVIEW.EXE).

The second filter, for a file with a **.ICO** extension (Icon), also provides a program path and file name (ICONEDIT.EXE).

```
PMWP_ASSOC_FILTER
       *.MET
         D:\OS!2 2.0 DESKTOP\OS!2_System\Productivity\WPProgram:A8134.
         D:\OS2\APPS\WPPROGRAMFILE:FPICVIEW.EXE.
       *.ico
         D:\OS2\APPS\WPPROGRAMFILE:FICONEDIT.EXE
```

Figure 53. Association Filters for File Extensions

The association type uses the same two formats as the filter; we can see the program reference 8134 again for metafiles and ICONEDIT.EXE for bitmaps in the figure below. File types are specified for each data file in its Settings view. We can see five file types below:

- Metafile
- Plain text
- OS/2 command file
- Executable
- Bitmap.

```
PMWP_ASSOC_TYPE
        Metafile
                D:\OS!2 2.0 DESKTOP\OS!2_System\Productivity\WPProgram:A8134
        Plain Text
                D:\OS!2 2.0 DESKTOP\OS!2_System\Productivity\WPProgram:A7660
        OS/2 Command File
                D:\OS!2 2.0 DESKTOP\OS!2_System\Productivity\WPProgram:A7660
        Executable
                0000  00
        Bitmap
                D:\OS2\APPS\WPPROGRAMFILE:FICONEDIT.EXE
```

Figure 54. Association Filters for File Types

When a user &dcs. on an icon for an object belonging to any of these file types, the program associated with it is started. Note that there is no association for executable files since they themselves are started. The file name association takes precedence over the file type association if both are specified for an object.

8.6 Multiple Instances of Objects

Within the Workplace Shell there are many folder and printer icons; one for each folder or attached printer in the system. There are two issues here: When we see multiple printers icons, do they refer to the same printer or to different printers, and do they share the same code?

Multiple copies of folders refer to different folders but they all share the same program. They are instances of the same folder class. Multiple copies of printers may look similar but may actually be instances of different printer classes.

To understand this, you have to understand something about the structure of the Workplace Shell and the System Object Model (SOM) which it is built on. Through inheritance, the System Object Model provides folders, work areas and other types of container objects which share their common code so the programmer only has to code the differences. SOM allows multiple instances of each class to be created and manages the memory, pointers, etc. for each of them. It also, through inheritance, allows instances of different classes to be created which look the same and share a significant degree of common code.

What happens when multiple instances of a data file are created depends on the technique used to create the object. If you perform a copy, then a new file and file EAs are created in the directory corresponding to the folder which the file was created in. The file name and all the other details are copied verbatim. Here, each copy is a new instance of the *WPDataFile* class and can be modified without affecting the original file.

If you create the copies in the same directory as the original, however, the operating system settings determine what happens when you try to copy a file into a directory which already contains a file of that name. As a default, the WPS will add a suffix to the file name and icon descriptive text; for the first copy a "1" is added, for the second a "2," and so on. The same thing happens if you create a copy in a different directory but then drag it back into the original directory while the original file is still there. This mechanism prevents you from accidentally

overwriting copies of your work, but it can be frustrating when you are trying to replace an out-of-date version of a file.

You can also make multiple instances of the same file using the shadow copy facility. Here, the shadow copy is an abstract object, stored as a HOBJECT pointer in OS2.INI which points back to the original file. In both cases, we have multiple instances of icons, but what the icons represent are instances of different classes with different characteristics. Here, when you modify the copy you are also affecting the original file.

8.7 Summary

The Workplace Shell is implemented using the System Object Model. Many of the characteristics of the Workplace Shell, such as folders and icons, are implemented as System Object Model classes. This method of implementation allows an application developer to use such classes and inherit characteristics and behaviors from them.

The ability to inherit standard behaviors from existing object classes can significantly simplify the task of developing a new object class, since only the differing behaviors must be explicitly defined; all others may simply be inherited from the parent class.

The WPS implementation also enables multiple instances of each class to be created, as well as multiple copies of any specific instance. WPS classes include data files, programs and shadow copies. Each class has its own methods for instantiating and storing objects of that class and use a variety of techniques for recording the attributes and data for each object.

The following table summarizes persistence in the Workplace Shell:

Type of Object	Object Location	Contents	Settings	Program Options
Data file	File	File	File EAs	
Program file	File	File	File EAs	Set by program
Shadow	OS2.INI	Original file	Original file	
Prog. ref.	OS2.INI	Original file	OS2.INI	Original program
Folder	Directory	Directory, OS2.INI	Directory EAs	

Table 2. Workplace Shell Object Persistence Summary

A data file stores its data in files within a disk directory and its attributes in Extended Attributes (EAs) associated with that file. A folder stores its contents in a disk directory and its attributes in EAs associated with that directory.

Program files are stored on disk but access to them via the WPS is controlled by pointers in the OS2.INI file. Shadow copies do not physically exist, they are simply pointers to the original data file which are held in the OS2.INI file. These pointers are also stored by folders in a section of the OS2.INI file. The position of their icons is held in the directory EAs for that folder.

Appendix A. Using REXX in OS/2 V2.0

REXX provides access to several OS/2 APIs via the the *RexxUtil* functions which have been added to OS/2 V2.0. To use these new functions in a REXX program, SysLoadFuncs must be called to automatically load all RexxUtil functions as follows:

```
/* Rexx program that uses RexxUtil functions */
call RxFuncAdd 'SysLoadFuncs', 'RexxUtil', 'SysLoadFuncs'
call SysLoadFuncs
```

RexxUtil functions are prefixed with "Sys." For a complete list of Sys* functions please refer to the *OS/2 2.0 Programming Guide Volume II*. In WPS development the following Sys* commands are most frequently used:

- SysRegisterObjectClass
- SysDeregisterObjectClass
- SysCreateObject.

The following syntax descriptions should help explain what these REXX functions do:

```
Function: SysRegisterObjectClass
Purpose:        Register a new object class definition to the system.
Syntax:  result = SysRegisterObjectClass(classname, modulename)

classname       The name of the new object class.
modulename      The name of the module containing the object definition.
result          The return code from WinRegisterObjectClass.
                This returns 1 (TRUE) if the class was registered
                or 0 (FALSE) if the new class was not registered.


Function: SysDeregisterObjectClass
Purpose:        Deregister an object class definition from the system.
Syntax:  result = SysDeregisterObjectClass(classname)

classname       The name of the object class to deregister.
result          The return code from WinDeregisterObjectClass.
                This returns 1 (TRUE) if the class was deregistered
                or 0 (FALSE) if the class was not deregistered.


Function: SysCreateObject
Purpose:        Create a new instance of an object class.
Syntax:  result = SysCreateObject(classname, title, location <,setup>)
classname       The name of the object class.
title           The object title.
location        The object location.  This can be specified as either a
                descriptive path (for example,
                OS/2 System Folder\System Configuration)
                or a file system path (for example, C:\bin\mytools).
setup           A WinCreateObject setup string.
result          The return code from WinCreateObject.
                This returns 1 (TRUE) if the object was created
                or 0 (FALSE) if the object was not created.
```

Appendix B. CUA Conformance in the Workplace Shell

Some differences exist between the CUA architecture and the Workplace Shell implementation in OS/2 V2.0. Some of these differences may force applications to diverge from the CUA architecture, either because of the effort required to override OS/2, or because of the negative impact to system consistency or customization if the CUA guidelines are followed.

This is of particular importance for CUA *Fundamental* items, which provide the foundation for many of the enhancements to CUA described in the "CUA Vision" demonstration and video. Some of the discrepancies between OS/2 and CUA are beyond the control of an individual application; under these circumstances the application has no choice but to use the OS/2 conventions.

However, in other situations it is quite easy for an application to comply with the CUA guidelines rather than following the example of OS/2. To aid in this determination, an assessment of each OS/2 item has been made, and the results are summarized below in Table 3 on page 134.

The same assessment has been made for non-fundamental (*Recommended*) items in Table 4 on page 135. Again, where the OS/2 implementation does not force an application to differ from the CUA architecture, the developer might simply use the OS/2 Workplace Shell as an example. In such cases the product could just as easily follow the CUA recommendations and should do so for improved consistency and ease of use.

OS/2 Version 2.0 - Volume 4: Application Development provides several examples of how to modify WPS objects and their behaviors, including subclassing folders and changing their icon description. These examples should prove useful to programmers who wish to implement CUA-conforming behaviors in their OS/2 V2.0 applications.

The relevant page in the *IBM Systems Application Architecture CUA Advanced Interface Design Reference* is indicated in the Reference Page column in both tables, together with an indication of which entry on that page is being applied. Each page of the *IBM Systems Application Architecture CUA Advanced Interface Design Reference* contains a definition, a "When to use" and a "Guidelines" section. The numbering scheme used refers to the item number within a section; W is "When to use" and G is "Guidelines," so 216/G2 is read as Page 216, second item in the Guidelines section.

B.1 Fundamental Items

Table 3 (Page 1 of 2). Fundamental Items

Description Problem Statement Recommendations/Alternatives	Reference Page[1]
There are several areas where OS/2 V2.0 mouse behavior is inconsistent with that described in CUA 91. In general, applications should follow OS/2 for UI consistency since consistency between OS/2 and application is of primary importance to users. However, they must use logical messages rather than responding to particular mouse buttons, so that user customization, or a future change in OS/2, will automatically be reflected in the application.	153/G13, 184/G2, 257/G10
OS/2 does not contain visible menu bars on object container windows. Lack of menu bars is likely to cause coexistence and migration difficulties. Testing also indicates that users prefer to have menu bars present. Applications have control over the window frame and should therefore provide menu bars.	141/G2
OS/2 has merged the system menu icon and title bar icon along with some of their functions. Title bar icons, when provided, must behave as a manipulable item which behaves the same as the object's icon; the merged icon in OS/2 does not exhibit this behavior. One problem with this is that the consistency of the OO user interface, which requires clear separation of model, view and controller functions, is reduced. Many applications will choose not to add a separate title bar icon because the title bar will then present two identical icons side-by-side; they look the same but act differently. Unless an application has subclassed WPS objects, such as Folders, they have control over the System Menu contents and should conform to CUA 91. If a program provides a title bar icon then it must conform to CUA rules.	113/G3, 253/W1 and Definition
OS/2 does not allow users to change the view in a window. The *IBM Systems Application Architecture CUA Advanced Interface Design Reference* requires applications to support changing the view in a window, so that users are not forced to open another window onto the same object, Applications have control over views and should follow the CUA architecture recommendations.	268/W1
OS/2 has implemented "conditional" cascade menus which behave differently from CUA rules on cascade menus. Selecting a cascading choice can cause a default action to occur unless the user clicks on the arrow portion of the choice. The CUA architecture recommends avoiding conditional cascade menus.	36/G1
OS/2 currently does not display scroll bars in windows if all objects are displayed. CUA maintains that scroll bars must always be displayed, following unavailable-state emphasis guidelines; this presents the user with a visual cue that the object extends beyond the window boundary. If the application has implemented its own container control, then it should conform to the CUA guidelines.	216/W1, W2, G2
DOS Session Settings uses a scroll bar to set numeric values for such things as FILES, FCBS, RMSIZE, etc. This is inconsistent with the behavior of scroll bars in windows. Both slider or spin button controls are available in OS/2 V2.0 and should be used to set numeric values; scroll bars should only be used for scrolling.	216/G3

Table 3 (Page 2 of 2). Fundamental Items	
Description **Problem Statement** **Recommendations/Alternatives**	**Reference Page[1]**
Message windows must have title bars and correct window frames. Some OS/2 message windows lack these features. Since applications have control over the generation of these windows they should conform to CUA guidelines.	272/G1
Note: [1] W is "When to use" and G is "Guidelines," so 216/G2 is read as Page 216, second item in the Guidelines section.	

B.2 Recommended Items

Table 4. Recommended Items		
Description **Problem Statement** **Recommendations/Alternatives**	**Reference Page[1]**	**Dependency on OS/2**
OS/2 does not provide a separate title bar icon. These are recommended by the CUA architecture. Many programs will wish to wait until this is supported in OS/2.	272/G2, 113/G3	YES
Changing a folder into a work area and vice versa should result in an immediate change in visual appearance of the object icon. If a program implements its own containers using the container control, it should adhere to CUA guidelines, but most applications will probably prefer to use the OS/2 facilities.	114/G2	YES
OS/2 does not always provide progress indicators during lengthy operations which indicate percent completion, and allow users to halt system execution. Applications should provide these progress indicators. OS/2 does provide controls for implementing progress indicators.	191/W1	NO
OS/2 does not provide and use an information area in its windows. These are very useful in providing timely feedback to users. Adherence to CUA is required.	119/W1	NO
OS/2 does not display a count of objects in containers. The container control keeps track of count (CM_QUERYCNRINFO). However, OS/2 does not display it on WPS containers. If a program implements its own containers using the container control, it should adhere to CUA guidelines, but most applications will probably prefer to use the OS/2 facilities.	51/G7	YES
Note: [1] W is "When to use" and G is "Guidelines," so 216/G2 is read as Page 216, second item in the Guidelines section.		

B.3 Other Items

In a shell with as many objects as the Workplace Shell, it is almost inevitable that some of the windows may contain elements which do not comply with CUA guidelines. Application developers should not assume that these elements and their behaviors are sanctioned by CUA; they should continue to implement objects and behaviors which conform to the rules laid out in the *IBM Systems Application Architecture CUA Advanced Interface Design Reference.*

This section provides a few examples of non-compliant behaviors in the following areas:

- Navigation
- Emphasis
- Mnemonics
- Push buttons
- Miscellaneous.

B.3.1 Navigation

Notebook	Up Arrow key moves the cursor from the notebook page to notebook tab
Glossary Settings	On Properties page, Tab key moves the cursor within push button field
Mouse Settings	On Mappings page, Arrow key moves the cursor between radio button field and checkbox field.

B.3.2 Emphasis

Dialog Editor	Help menu choice and all the choices in the pull-down are displayed with unavailable-state emphasis
Clipboard Viewer	In the "File" menu, Import and Export choices are never available yet they are displayed with unavailable-state emphasis
Font Palette	Target emphasis is not displayed during direct manipulation operations.

B.3.3 Mnemonics

Shredder	Mnemonic is missing from Refresh choice in pop-up menu
Format	In Progress window, mnemonic is missing from Stop push button
Menu Settings	Incorrect terminology and no mnemonic for predefined push button
DOS Settings	Mnemonics are incorrectly assigned to Help and Cancel push buttons.

B.3.4 Push Buttons

Glossary List	Push buttons are not justified from the left
Format	In Progress window, Close push button is missing
Device Driver Install	Exit push button performs Close function.

B.3.5 Miscellaneous

Edit Font	Pressing Enter key does not cause default action to begin in Action window
Notebook	Cursor is not visible on notebook page when focus is moved from tab to page using the keyboard
System Error	Message window is missing the system menu icon and pull-down
Shell	Alt+Tab key switches between unassociated windows
Clipboard Viewer	Exit is redundant in the File menu; Close in the system menu performs this same function

Glossary

API. Application Programming Interface; term used to describe the set of functions by which an application may gain access to operating system services.

application-controlled viewport. Viewport within a help window or online document window, where the display of information within that viewport is controlled by an application, which is specified by the developer of the source information. Application-controlled viewports may be used to display image, video or other types of information under the control of the Information Presentation Facility. See also IPF-controlled viewport.

bit. A binary digit, which may be either zero or one. Bits are represented within a computing device by the presence or absence of an electrical or magnetic pulse at a particular point, indicating a one or a zero respectively.

Boot Manager. Boot Manager; feature of OS/2 Version 2.0 which allows multiple partitions to exist on fixed disks in the same machine, with a separate operating system on each partition. At boot time, the user may select the desired operating system with which to start the machine.

byte. A logical data unit composed of eight binary digits (bits).

Common User Access. Component of IBM Systems Application Architecture, which defines standards and guidelines for user interfaces in both character-based and GUI applications.

compatability region. In the OS/2 Version 2.0 flat memory model, the address region below 512MB, which may be addressed by a 16-bit application using the 16:16 addressing scheme and tiled local descriptor tables. Under OS/2 Version 2.0, this region is equivalent in size to the process address space.

container object. An object in the SAA CUA Workplace Environment which allows logical grouping of objects in a manner determined by the user. A container object may contain work items, physical devices and/or other container objects. A new class of control window is implemented under OS/2 Version 2.0 to facilitate the creation and manipulation of containers.

context menu. A menu associated with an object or view, which appears when the user moves the mouse cursor over the object or view and clicks mouse button 1. The context menu allows actions to be performed upon the object or view, in a similar manner to that available with a menu bar under OS/2 Version 1.3.

CUA. See Common User Access.

DDE. Dynamic Data Exchange; interprocess communication protocol used by applications to define dynamic links. Information updated in one application is automatically reflected in other applications linked to the first application via DDE.

desktop. In the context of the SAA CUA Workplace Environment and the Workplace Shell, the background of the computer display, which represents an electronic analogy of the user's work environment. Icons, representing objects to be manipulated, are moved on the desktop in order to perform work tasks, in a style known as direct manipulation.

direct manipulation. Term used to describe a style of interface whereby icons representing objects or work items are moved on the desktop to perform operations by dropping the icons over other icons representing physcial devices such as printers or conceptual devices such as an "out basket" in an electronic mail system. Also known as drag and drop manipulation.

DLL. Dynamic link library; application module containing routines and/or resources, which is dynamically linked with its parent application at load time or runtime rather than during the linkage editing process. The use of DLLs enables decoupling of application routines and resources from the parent program, enhancing code independence, facilitating maintenance and reducing resident memory consumption.

drag and drop manipulation. See direct manipulation.

Extended Attributes. Information which may be associated with a file under OS/2 Version 1.2 or above (including Version 2.0), to indicate various properties of that file. Extended attributes are available with both the FAT and HPFS file systems. An application may define extended attributes for files which it creates, and may update the extended attributes of files upon which it operates. A number of standard extended attributes are defined by the operating system for commonly-used information.

FAT. File Allocation Table; term used to describe the file system implemented by DOS and OS/2. This file system uses a file allocation table to contain the physical sector addresses of all files on the disk. The FAT file system is supported by OS/2 Version 2.0, along with the newer HPFS and other installable file systems.

flat memory model. Conceptual view of real memory implemented by OS/2 Version 2.0, where the operating system regards memory as a single linear address range of up to 4GB.

folder. See container object.

general protection exception. Operating system error which occurs when an application attempts to access memory in a page which has not been allocated to that process. OS/2 Version 2.0 allows an application to trap a general protection exception using an exception handler registered with the operating system. If an exception handler is not registered, the operating system will terminate the application as a result of a general protection exception. Also known as a Trap 000D.

guard page. Page within a memory object, for which the PAG_GUARD attribute has been specified. Any attempt by the application to reference memory within the guard page results in a guard page exception.

guard page exception. Operating system warning condition which occurs when an application accesses memory within a page which has been declared as a guard page. This exception may be trapped using an exception handler registered by the application in order to handle such occurrances. The typical processing performed by the exception handler is to commit more memory within the memory object. If an exception handler is not registered, the operating system's default handler commits the next available page within the memory object and sets its attribute to PAG_GUARD.

GUI. Graphical User Interface; term used to describe a user interface which typically uses windows and graphical representation to allow an application to interact with the end user. Examples of such interfaces include OS/2 Presentation Manager and Microsoft Windows.

HPFS. High Performance File System; file system first implemented under OS/2 Version 1.2, offering enhanced performance over the original FAT file system implemented in DOS and prior versions of OS/2. HPFS is an optional installation item under OS/2 Version 2.0; the FAT system may also be used to retain compatibility with DOS.

hypergraphics. Under the Information Presentation Facility, a portion of a bitmap displayed in a help panel or online document, which may be selected by the end user. Selecting such an item causes an event to occur, such as the display of another help panel, a popup window, the dispatch of a message to the parent application or the start of a new application. See also hypertext.

hypertext. In the Information Presentation Facility, a word or phrase in a help panel or online document, which may be selected by the end user. Selecting such an item causes an event to occur, such as the display of another help panel, a context menu, the dispatch of a message to the parent application or the start of a new application. See also hypergraphics.

icon. A graphical image on a computer display, which represents an object such as a file or a physical device such as a printer or plotter. In the SAA CUA Workplace Model, icons are used to electronically represent objects in the user's work environment, which are manipulated by moving icons on the desktop.

Information Presentation Facility. Facility provided by OS/2 Presentation Manager which allows applications to create embeddded, context-sensitive help windows for their windows and dialog boxes. Features include built-in indexing and hypertext. Under OS/2 Version 2.0, the Information Presentation Facility is enhanced to include the ability to display hypergraphics.

IPF. See Information Presentation Facility.

IPF-controlled viewport. Viewport within a help window or online document window, where the formatting and display of information within that window is controlled by the Information Presentation Facility. This is the default case for information displayed using the Information Presentation Facility. See also application-controlled viewport.

Initial Program Load. Term used to describe the process of loading a program (operating system) into memory when a machine is switched on. Also known as "boot," a reference to "lifting oneself up by ones bootstraps."

IPL. See Initial Program Load.

MB. Megabyte; 1024 kilobytes, or 1024 x 1024 bytes.

memory object. Logical unit of memory requested by an application, which forms the granular unit of memory manipulation from the application viewpoint. A memory object may be up to 512MB in size under OS/2 Version 2.0.

Multiple Virtual DOS Machines. Feature of OS/2 Version 2.0 which enables multiple DOS applications to execute concurrently in fullscreen or windowed mode under OS/2 Version 2.0, in conjunction with other 16-bit or 32-bit applications, with full preemptive multitasking and memory protection between tasks. See also virtual DOS machine.

MVDM. See Multiple Virtual DOS Machines

notebook control. New class of control window implemented under OS/2 Version 2.0, which provides for the display and navigation of complex user dialogs involving multiple related dialog boxes.

NULL. A binary zero. In C programming terms, NULL is typically used to refer to a pointer which is set to the binary zero value.

object. A physical entity such as a printer or fixed disk device, or a logical entity such as file or docu-

ment, which is manipulated within the Workplace Shell under OS/2 Version 2.0, in order to perform a work task. Objects are typically represented within the Workplace Shell using icons.

page. Granular unit for memory management using the 80386 and 80486 processors. A page is a 4KB contiguous unit of memory, which the processor manipulates as a single entity for the purpose of memory manipulation and swapping.

page fault exception. Operating system error which occurs when an application attempts to access memory which has been allocated but not committed. This exception may be trapped by an application using an exception handler registered with the operating system. If an exception handler is not registered, the operating system's default handler will terminate the application as a result of a page fault exception. Also known as a Trap 000E.

pop-up menu. See context menu.

printer object. In the Workplace Shell, an icon on the desktop which represents a printer device to which output may be directed. Objects such as files or documents are printed by moving their icons over the printer object and releasing the mouse button, thereby dropping the object onto the printer.

protected mode. Mode of operation for the Intel 80286 and 80386/80486 processors, whereby the address space is expanded to 16MB (80286) or 4GB (80386/80486), and memory references are translated via segment selector and offset, enabling full memory protection between processes executing in the system. With the 80386 and 80486, paging is available in protected mode.

RAM. Random Access Memory; term used to describe memory which may be dynamically read and written by a processor or other device during system operatons. RAM is typically used to store program instructions and data which not being operated upon by the processor at the current moment in time, but which are required for the logical unit of work currently being carried out.

real mode. Default mode of operation for the Intel 80286 and 80386/80486 processors, and the only mode of operation for the 8086 processor. In real mode, the processor acts as a 16-bit device, its physical memory address space is limited to 1MB, and memory references translate directly to physical addresses. With the 80386 and 80486, paging is not supported in real mode.

reference link. Mechanism by which a shadow object is associated with the "real" object to which it pertains. See also shadow object.

ROM. Read-Only Memory; term used to describe memory which may be read, but not written to, during system operations. ROM is typically used to store basic hardware initialization instructions, BIOS or self-testing code, which is required to be available prior to accessing the disk subsystem.

SAA. See Systems Application Architecture.

segment. Unit of memory addressable by the Intel 80x86 processors. With the 8086 and 80286 processors, a sgement may be from 16 bytes to 64KB in size. With the 80386 and 80486 processors, a segment may be up to 4GB in size.

segment selector. Field which specifies the base address of a memory segment when using the segmented memory model. The selector is 16 bits in length on an 80286 processor, and 32 bits in length on an 80386 or 80486 processor.

semaphore. Construct used under OS/2 Version 2.0. and previous versions of OS/2 to enable synchronization between processes and between threads in the same process. OS/2 Version 2.0 provides enhanced semaphore facilities over previous versions.

service layer. Executable code which performs the operating system function requested by an application using an API.

shadow object. An object on the desktop or in a folder, which actually references another object. Shadow objects allow an object such as a device to be defined in multiple locations. For example, a high-quality printer object may be defined in both a *Word Processing folder* and a *Spreadsheets* folder. A shadow object is associated with a "real" object via a reference link.

shredder object. Object on the Presentation Manager desktop which allows an object such as a file or document to be erased from the system, by dropping the object onto the shredder object's icon.

slider control. New class of control window implemented under OS/2 Version 2.0, for use when a particular property or value should be set by analogue rather than exact digital means.

SOM. See System Object Model

sparse object. Memory object for which a linear address range has been reserved, but for which no physical memory has yet been committed. This capability is used to reserve storage in the process address space for use by an application, without causing an adverse impact on system performance by requesting large amounts of physical memory.

Systems Application Architecture. Set of rules, standards and guidelines introduced by IBM in 1987 to facilitate ease-of-use of applications, and portability between different operating system environments. Systems Application Architecture consists of three

components; Common User Access, Common Programming Interfaces and Common Communications Support.

System Object Model. Language-independent, object-oriented mechanism in which the Workplace Shell is written. It consists of a specification for language-independent message passing and inheritance for objects, some base classes from which the WPS class hierarchy is derived, and language bindings for C.

thunk. Term used to describe a routine which performs address conversion, stack and structure realignment, etc. Thunking is used to pass control between 16-bit and 32-bit application modules.

Trap 000D. See general protection exception.

Trap 000E. See page fault exception.

value set. New class of control window implemented under OS/2 Version 2.0, for use when selecting an item from a finite set of mutually exclusive choices. Similar in function to a radio button, but has the added flexibility of being able to display graphical items such as color patches, icons or bitmaps.

view. Particular visual representation of an object under the Workplace Shell. An object may have several available views; for example, and icon view depicts the objects as an icon, whereas a settings view enables the user to set properties and define the appearance of other views.

viewport. Under the Information Presentation Facility, a portion of a help window or online document window which may be separately manipulated. The use of multiple viewports in a window enables the display or different types of information in the same window, with separate formatting and scrolling. Viewports may be either simple or complex, and may be IPF-controlled or application-controlled.

VDM. See Virtual DOS Machine

Virtual DOS Machine. A protected mode process under OS/2 Version 2.0 which emulates a DOS operating system environment, such that DOS applications executing within the virtual machine operate exactly as if they were running under DOS. virtual DOS machines support both text and graphics applications, and make use of the virtual 8086 mode of the 80386 and 80486 processors.

Workplace Environment. User interface model for GUI systems under IBM 1991 Systems Application Architecture Common User Access. The Workplace Environment defines guidelines for an interface where icons are manipulated using a mouse or keyboard, to perform work tasks electronically in a way analagous to that in which they would be performed manually. Such an interface facilitates user training and allows greater productivity through its intuitive nature.

Workplace Shell. Object-oriented user shell implemented by Presentation Manager in OS/2 Version 2.0. The Workplace Shell uses icons to represent objects such as files or devices, and allows the user to perform work tasks by directly manipulating these icons on the desktop with the mouse or keyboard.

WPS. See Workplace Shell

0:32. Term used to describe the addressing scheme used for the 32-bit flat memory model, where a memory address is expressed as a 32-bit offset within the linear address range.

16:16. Term used to describe the addressing scheme used for the 16-bit segmented memory model, where a memory address is expressed as a 16-bit segment selector, and a 16-bit offset within that segment.

16-bit. Term used to describe an application which uses the 16:16 addressing scheme implemented under DOS and previous versions of OS/2. In fact, such applications use a 24-bit address since the segment selector and offset are normally overlapped. Such applicaitions typically use the 16-bit instruction set implemented under the Intel 80286 processor.

32-bit. Term used to describe an application which uses the 0:32 addressing scheme implemented under OS/2 Version 2.0. Such applications may make full use of the 80386 instruction set.

80386. Intel 80386 microprocessor; the 32-bit processor upon which the OS/2 Version 2.0 operating system is based.

Index

A

Abstract objects 112
  programs 119
  shadows 119
Active window 13
Advanced operation of the Workplace Shell 60
API
  direct manipulation 121
  dynamic data facility 98
  LAN Independent Shell 50
  PM access to WPS 120
  PM enhancements 97
  PM Graphics Programming Interface 98
  PM printing functions 98
  SOM/WPS 120
Application model
  Presentation Manager 91
  Workplace Shell 99
Application structure 103
Applications
  integrating with WPS 103
  migrating 104
Arranging the desktop
  arranging folders and objects 40
Association 105
  adding new file types 107
  ASSOCTABLE 71, 105, 107
  by file name 121
  by file name and extension 62, 63
  by file type 62, 63, 121

B

Backup and restore
  backup programs
    PMTAPE software 76
    SY-TOS Plus software 76
  critical system files
    CONFIG.SYS 74
    effect of restoring OS2.INI 76
    how to backup OS2.INI 74
    OS2.INI 74
    OS2SYS.INI 74
    restoring OS2.INI 75
  restoring OS2.INI
    booting from diskette 75
    system install from Alt-F1 76
Base classes, WPS 112
Basic operation of the Workplace Shell 56

C

Changing an objects icon 62
Check box 15
Class
  definition file 101
  deregistering 82
  implementing a new, 102
  registering 82, 99
Classes
  hierarchy 101
  using 101
Client Area
  use of control windows 15
Clipboard
  definition 17
Combo box 15
Container control
  definition 6
  details view 22
  flowed name view 22
  icon view 22
  name view 22
  text view 22
  tree view 23
Container object 42
Context menu 7
  modifying 60
  mouse button 1 56
  popping up 56
Control windows
  check box 15
  combo box 15
  container 6, 21
  entry field 15
  list box 15
  notebook 6, 23
  progress indicator 27
  push button 16
  radio button 15
  slider 6, 25
  spin button 16
  use in client area 15
  use in dialog box 15
  value set 6, 26
Creating a new object 57
Creating a shadow copy 58
Creating a startup environment 59
Creating new file types 84
CUA perspective on WPS 34
Customizing an object 60
Customizing the desktop
  colors 39
  country 40
  fonts 39
  keyboard 39
  mouse 39

Customizing the desktop *(continued)*
sound 40
system clock 40
Customizing the Workplace Shell 80
Customizing the Workplace Shell objects 38

D

Data object 42
DDF
see Dynamic Data Facility
Desktop background 36
Desktop placement 69
Details view, container 22
Device context 97
Device object 42
Dialog box
definition 13
modal 13
modeless 13
use of control windows 15
Direct manipulation 120, 121
printing by 107
rendering mechanisms 104
Directory
EA contents 124
WPS structure 113
Disk partitions
Desktop placement 69
FAT format 68
HPFS format 68
print spooler placement 69
Display window 91
DM_DISCARDOBJECT message 123
DM_DRAGOVER message 122
DM_PRINTOBJECT message 123
Drag/Drop
see Direct Manipulation
DRM_OS2FILE message 123
Dynamic Data Facility 98

E

Entry field 15
Existing applications 2
Experienced users 3
Extended Attributes 115
directory 124
EA_DATA. SF file 124
file 124, 125
in FAT file systems 70
in HPFS file systems 70
support by DOS programs 70
WP_ROOT. SF file 124
WP_SHARE. SF file 124

F

Facilities, WPS 120
File 62
association with a program 121
associations 62
EA contents 125
file associations 120
settings 117
shadow copies 119
views 117
file contents 115
File dialog, "Open" 28
File dialog, "Save as" 28
File System objects 112, 117
File transfer to a host session
context menu 82
Icon 81
File type
adding new types 107
associating data files with 63
Finding open windows 57
Flowed name view, container 22
Folder 21, 37, 42
implementation 115
local 78
population 116
settings 116
Font dialog 29
Frame Area
menu bar 11
minimize icon 11
sizing border 11
title bar 11
Frame controls
maximize icon 12
menu bar 11
minimize icon 12
restore icon 12
scroll bars 12
sizing border 11
small icon 12
system menu 12
title bar 11
title bar icon 12

G

Graphics Programming Interface 98
see Presentation Manager

H

Helper macros 99
Hypergraphics 30
Hypertext 30, 31

I

Icon Editor 17
Icon view, container 22
Icons 11, 16
Implementation of the Workplace Shell 111
Information Presentation Facility
  appearance 30
  enhancements 6
  external links 30
  hypertext 31
  split screen support 30
Inheritance 100
Input focus 10, 13
Installation 67, 71
  for an inexperienced user 84
  programs 71
  restricted users system 86
  Workplace Shell, considerations 67
Interprocess communications 103
IPF
  See Information Presentation Facility

L

LAN Independent Shell
  API 50
  configuration files 50
  description 47
  local folders 49, 78
  making shadow copies 49
  moving objects 49
  multiple network support 49
  multiple server support 49
  Network folder 47, 48
  Network icon 47
  Novell Netware support 50
  organization of a LAN workplace 76
  OS/2 LAN Server support 50
  printer support 49
  Public Applications folder 47
  servers 47
  shadow folder 77
  shadow folders 49
  shared access to resources 49
  universal naming convention 49
Limiting a users access to settings 82
List box 15
Local printing 59

M

Major tabs, notebook 24
MAKEINI command 84
Making OS/2 look like OS/2 Version 1.3 64
Maximize icon 12
Menu bar
  definition 11
  pulldown menu 11
Message box
  definition 14
Messages
  classes 96
  definition 94
  processing 96
  queues 94
Minimize icon 12
Mixed model programming 7
Modifying an objects menus 60

N

Name view, container 22
Navigation 53
  accelerators 54
  keyboard 53
  mouse 53
Network folder 48
Notebook control
  appearance 23
  definition 6
  major tabs 24
  scroll arrows 25
  top page 24
Novell Netware 47, 49, 50, 70

O

Object Interface Definition Language 100
Object window 91
Object-oriented programming 100
  model-view design 103
Objects 21, 44
  abstract 112, 119
  classes 41
  creating a shadow copy 58
  device 42
  direct manipulation 55
  drag and drop 55
  File System 112, 117
  manipulation techniques 54, 55
    traditional approach 54
  registration 120
  shredder 44
  transient 112, 120
  types 41, 112
  versus applications 99
  WPS versus SOM 5
  WPS, multiple instances of 120, 129
  WPS, standard 41
OIDL
  see Object Interface Definition Language
Opening a window 56
Organization of a LAN workplace 76
OS/2 Version 1.3, making OS/2 look like 64
OS2.INI 115, 119, 127
  abstract objects references 127
  adding file types 108

OS2.INI *(continued)*
association filters 128
association types 128
folder contents 127
MAKEINI command 84
modifying 84
removing WPS items 84
running programs 127
structure and contents 126

P

Page buttons 24
Performance considerations 79
restarting programs 79
using work areas 79
Persistence 123
Pop-up menu
see Context menu
Presentation Manager
API enhancements 97
definition 9
device context 97
dynamic data facility 98
enhancements in V2.0 5
file dialog 98
font dialog 98
functions removed in V2.0 97
GPI 98
graphics functions 98
helper macros 99
icons 11, 16
message queues 94
messages 94
new window classes 98
objectives 3
presentation space 97
printing functions 98
programming environment 7
what is it? 3
windows 91
workplace functions 98
Presentation space 97
Prevent programs restarting at IPL 80
Printing
from DOS programs 59
local 59
PM APIs 98
printer objects 58
remote 59
spooler placement 69
Problem determination
Desktop malfunction
extra printers 73
multiple instances 73
system hangs 73
HPFS 73
OS2.INI 73
Shutdown 73
Program 108
ASSOCTABLE 71
installing 71
registration 108
Programming environment
16-bit application compatibility 7
context menu 7
memory management 7
mixed model programming 7
standard dialogs 7
Programs, adding to a data file context menu 64
Programs, associating filenames with 63
Progress indicator control
appearance 27
push button 28
Pulldown menu 11
Push button 16

R

Radio button 15
Remote printing 59
Rendering mechanisms 104
OS/2 file 104
print 104
Requirements, restricted user 85
Resource script file 105, 108
Restarting programs 79
Restore icon 12
Restricted user
installation 86
requirements 85
Restricting user access to functions 84
REXX utilities 80
administrators 82
notes on using with OS/2 V2.0 131
RexxxUtil functions 131
SysCreateObject. 131
SysDeregisterObjectClass 131
SysLoadFuncs 131
SysRegisterObjectClass 131

S

SAA CUA Workplace environment 34
Scroll bars 12, 26
Settings view 60, 62
shadow 77
Shadow copies
file 119
Shadow copy 58
creating 58
Shadow folder 77
Shell, user interface 4
Shredder object 44
Sizing border 11
Slider control
appearance 25
definition 6

Slider control *(continued)*
slider arm 26
slider shaft 26
Small icon 12
SOM
converting a PM program 108
see System Object Model
Spin button 16
Standard dialogs
definition 7
file dialogs 28
font dialog 28
Open 28
Save as 28
Startup folder sequence 59
Supporting the Workplace Shell 67
System menu 12
System Object Model 100, 112, 130
compiler 100
Systems Application Architecture
Common User Access
control windows 11, 15
graphical model 12
objectives 9
workplace environment 16

T

Templates
contents after installation 46
user perspective 46
Text view, container 22
Title bar 11
Training users on the WPS 79
Transient objects 112, 120
Tree view, container 23

U

UNC
Universal Naming Convention 49
User
requirements 85
restricted 85
User interface 9
Using the Workplace Shell 53
Utilities, REXX 80

V

Value set control
appearance 26
definition 6
Viewports
definition 30
Views 37, 56

W

Window 91
active window 13
child window 13
classes 93
client area 11, 12
clipboard 17
control window 15
definition (application view) 91
definition, user view 9
dialog box 13
display window 91
frame area 11
input focus 10, 13
menu bar 11
message box 14
minimize icon 11
object window 91
parent window 13
sizing border 11
title bar 11
window procedure 93
window words 93
Window list 57
customizing 57
Work area 42
creation 87
performance considerations 79
restricted user 87
using 79
Workplace Shell 33
abstract objects 112, 119
advanced operation 60
application environment 5
application structure 103
association 62
basic operation 56
common objects 41
container 42
context menu 56
creating a Shadow Object 58
creating folders 82
customizing for specific users 80
creating new file types 84
File transfer to a host session 81, 82
installing OS/2 for an inexperienced user 84
Limiting a users access to settings 82
prevent programs restarting at IPL 80
removing WPS objects 84
restricting user access to functions 84
Startup folder sequence 59
data objects 42
delete 58
desktop 36
desktop background 36
direct manipulation 120, 121
drives 43
EA_DATA. SF file 124

Workplace Shell *(continued)*
events/messages
delete 123
DM_DISCARDOBJECT message 123
DM_DRAGOVER message 122
DM_PRINTOBJECT message 123
DRM_OS2FILE message 123
file move/copy 123
print 123
Extended Attributes
directory 124
EA_DATA. SF 124
file 125
WP_ROOT. SF 124
WP_SHARE. SF 124
facilities 120
file associations 120, 121
File System objects 112, 117
folder population 116
folders 37, 42, 113, 115
icons 113
implementation 111, 112
installation considerations 67
Desktop placement 69
disk partitions 67
Extended Attributes 70
FAT partitions 68
HPFS partitions 68
planning the disk layout 67
print spooler placement 69
setting up programs 71
setting up programs and files 70
LAN independent shell 76
modifying OS2.INI 84
multiple instances of WPS objects 120, 129
multiple threads 111
navigation 53
object registration 120
object types 41, 112
objects 115
objects and menus 56
changing default view on "open" 61
modifying 60
OS2.INI structure and contents 126
performance considerations
restarting programs 79
using work areas 79
persistence 123
populating folders 82
print objects 45
program references 43
programming environment 7
reference books 43
relationship to file system 113
REXX utilities 80
RexxxUtil functions 131
SysCreateObject. 131
SysDeregisterObjectClass 131
SysLoadFuncs 131

Workplace Shell *(continued)*
REXX utilities *(continued)*
SysRegisterObjectClass 131
SAA CUA Workplace environment 34
shadow copies 43, 119
single process 111
support considerations 67
backing up OS2.INI 74
backup and restore 74
backup programs 76
critical system files 74
file EAs used by DOS programs 70
files outside the WPS directory structure 71
LAN independent shell 76
problem determination 73
restoring OS2.INI 75
symptoms of desktop malfunction 73
System Object Model 112, 130
transient objects 112, 120
user training 79
user's perspective 33
using 53
utilities, REXX 80
views 37, 56
what is it? 4
window list 57
work areas 42
WP_ROOT. SF file 124
WP_SHARE. SF file 124
WPS as an OS/2 program 111
WPS as an OS/2 program 111

Readers' Comments

OS/2 Version 2.0
Volume 3: Presentation Manager
and Workplace Shell

Publication No. GG24-3732-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

IBM®

Cut
Alor

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Center
Department 91J, Building 235-2
Internal Zip 4423
901 NORTHWEST 51ST STREET
BOCA RATON FL 33431-1328

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alor

Readers' Comments

OS/2 Version 2.0
Volume 3: Presentation Manager
and Workplace Shell

Publication No. GG24-3732-00

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Phone No.

IBM®

Cut
Alon

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Center
Department 91J, Building 235-2
Internal Zip 4423
901 NORTHWEST 51ST STREET
BOCA RATON FL 33431-1328

Fold and Tape | **Please do not staple** | Fold and Tape

Cut
Alo

GG24-3732-00

OS/2 Version 2.0 Volume 3: Presentation Manager and Workplace Shell

GG24-3732-00

PRINTED IN THE U.S.A.

IBM®

GG24-3732-00