



Macintosh. **MacWorkStation™**  
**Programmer's Guide**

**Production Draft**

June 19, 1989

**Apple Confidential**

Networking and Communications Publications

CommuniTree Group

415-441-3088

AppleLink: X0541

## **APPLE COMPUTER, INC.**

This manual is copyrighted by Apple or by Apple's suppliers, with all rights reserved. Under the copyright laws, this manual may not be copied, in whole or in part, without the written consent of Apple Computer, Inc. This exception does not allow copies to be made for others, whether or not sold, but all of the material purchased may be sold, given, or lent to another person. Under the law, copying includes translating into another language.

© Apple Computer, Inc., 1988  
20525 Mariani Avenue  
Cupertino, CA 95014  
(408) 996-1010

Apple, the Apple logo, AppleShare, AppleTalk, and Macintosh are registered trademarks of Apple Computer, Inc.

APDA, Finder, LocalTalk, MacWorkStation, MPW, MultiFinder, QuickDraw, and ResEdit are trademarks of Apple Computer, Inc.

DEC is a trademark of Digital Equipment Corporation.

Ethernet is a registered trademark of Xerox Corporation.

IBM is a registered trademark of International Business Machines Corporation.

MacDraw, MacPaint, and MacWrite are registered trademarks of Claris Corporation.

MasterCard is a registered trademark of MasterCard International, Inc.

Microsoft is a registered trademark of Microsoft Corporation.

VISA is a registered trademark of VISA International Service Association.

Simultaneously published in the United States and Canada.

## **Disclaimer of Warranty**

The manual and media are provided "as is," without warranty of any kind, either express or implied, including without limitation any warranty with respect to its merchantability or its fitness for any particular purpose. The entire risk as to the quality and performance of the manual and media is with you. Should the manual or media prove defective, you (and not Apple or an Apple-authorized representative) assume the entire cost of all necessary servicing, repair, or correction.

Apple does not warrant that the functions contained in the manual and media will meet your requirements or that the operation of the manual and media will be uninterrupted or error free or that defects in the manual and media will be corrected.

Some states do not allow the exclusion of implied warranties, so the above exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

# Contents

Figures and tables / vi

## **Preface About This Guide / ix**

Who should read this guide? / x

What this guide contains / x

Conventions used in this guide / xi

What you need / xi

The MacWorkStation Developer's Kit / xii

Development system documentation / xiii

## **1 What Is MacWorkStation? / 1**

The MacWorkStation-client system / 2

What MacWorkStation does / 3

Types of MacWorkStation-client systems / 4

Why a Macintosh interface? / 5

Human interface guidelines / 5

Three qualities of a good program / 6

Responsiveness / 6

Permissiveness / 6

Consistency / 6

General suggestions for application programs / 7

Modes / 7

Graphics and color / 8

## **2 Using MacWorkStation / 9**

Creating the MacWorkStation-client system / 10

Starting a MacWorkStation-client system / 11

The client application / 12

The event loop / 13

About the "Hello.C" sample application / 15

The "Hello.C" sample application / 16

The MacWorkStation document / 19

MacWorkStation-client communications /	20
Communication Command Language (CCL) scripts /	20
Communication modules /	23

### **3 Directors and Messages / 25**

MacWorkStation directors /	26
MacWorkStation messages /	28
The structure of a message /	29
Class /	30
Identifier /	30
Parameters /	30
Using aliases /	32
Examples /	32

### **4 Using Directors / 33**

About the "Bear Cal" application /	34
The Alert Director /	35
The Cursor Director /	36
The Dialog Director /	37
Dialog item types /	38
Control items /	38
A dialog box routine /	39
Clustering items /	41
The File Director /	42
The Graphics Director /	43
Coordinates in graphics windows /	43
Graphics commands /	44
The List Director /	46
The Menu Director /	48
The Process Director /	49
The Text Director /	49
The Window Director /	50
Window kinds /	50
Window shapes /	51
Positioning windows /	52
Window options /	52
The Exec Director /	55

**Appendix "Bear Cal" Program Source Code / 57**

**Glossary / 91**

# Figures and tables

## CHAPTER 1 What Is MacWorkStation? / 1

Figure 1-1 MacWorkStation message protocol / 2

Figure 1-2 A modal dialog box / 7

## CHAPTER 2 Using MacWorkStation / 9

Figure 2-1 The MacWorkStation-client system / 11

Figure 2-2 Screen created by a simple application

Figure 2-3 Choose-script dialog box / 20

Figure 2-4 Script-edit window with Command Help window / 21

Figure 2-5 New script name dialog box / 22

## CHAPTER 3 Directors And Messages / 25

Figure 3-1 Dialog box from the Bear Cal sample application

Figure 3-2 Macintosh menu example / 29

Table 3-1 Classes and directors / 30

## CHAPTER 4 Using Directors / 33

Figure 4-1 Alert created with the Alert Director / 35

Figure 4-2 Wristwatch pointer created with the Cursor Director / 36

Figure 4-3 Reservations dialog display with clusters / 40

Figure 4-4 Screen display created with the Graphics Director / 44

Figure 4-5 Flight schedule created with the List Director / 46

Figure 4-6 Pull-down menu example / 48

Figure 4-7 Window shapes / 51

Figure 4-8 List window created with the Window Director / 54

Table 4-1 Cursor resource IDs / 36

Table 4-2 Keywords and dialog item types for the MacWorkStation Dialog  
Director / 38

Table 4-3 Window kinds / 50

Table 4-4 Window options / 53





## Preface

## About This Guide

SINCE THE APPLE® MACINTOSH® computer was first introduced, many people have wanted to interact with Macintosh users from other computers, such as mainframes or other personal computers, through the human interface provided by the Macintosh computer. The MacWorkStation™ applicaton (MWS) makes this possibility a reality. Now, by using MWS, you can write applications that let Macintosh users communicate with another computer by means of windows, pull-down menus, dialog boxes, and other features of the Macintosh desktop interface. This guide introduces you to MWS and how it works with the other computer's application.

In explaining the various elements of a MacWorkStation system, this guide assumes that you are familiar with one or more programming languages and that you have had some experience using a Macintosh computer. You also need to be familiar with the other computer you will be using, and how to program it. ■

---

## Who should read this guide?

This guide is designed for programmers who are new to writing application programs that use the Macintosh desktop interface. If you haven't already done so, take some time to explore the Macintosh computer by using programs such as MacPaint, MacWrite, and MacDraw.

Notice the use of windows, menus, desk accessories, dialog boxes, and scroll bars. Become familiar with such terms as *click* and *drag*. Be sure you know the meaning of *mouse* and *icon*. Try actions such as cutting, pasting, and copying. If necessary, review the Macintosh owner's guide before you begin writing your own applications. If you want to learn more about the Macintosh interface, you will find it helpful to read some of the material recommended at the end of this preface.

Using this manual and the *MacWorkStation Programmer's Reference*, you will be able to write a variety of application programs that use the Macintosh desktop interface.

---

## What this guide contains

This guide is divided into four chapters and one appendix that contain the following information:

- Chapter 1, "What Is MacWorkStation?" describes how the MacWorkStation application works, discusses the Macintosh user interface and the elements in a good Macintosh application, and offers general suggestions for writing applications.
- Chapter 2, "Using MacWorkStation," tells how to create a MacWorkStation application. It presents a simple MacWorkStation application program that demonstrates basic concepts common to all applications.
- Chapter 3, "Directors and Messages," presents an overview of the various MacWorkStation components and how they interface with the Macintosh computer, and describes how MWS communicates with the application running on the other computer.
- Chapter 4, "Using Directors," examines parts of a sample MacWorkStation program written in C and shows how the various director commands are used.
- The appendix, "Bear Cal' Program Source Code," gives the complete listing of the program discussed in Chapter 4.

Each chapter builds on material from preceding chapters, so you should go through them in order.

## Conventions used in this guide

This guide uses typographic conventions to distinguish between different kinds of words and symbols.

- Terms in **boldface** are defined in the glossary at the end of this book.
- Elements of computer language are printed in a `fixed-width font`.
- Message parameters that are variables are printed in *italics*. These terms can be replaced by any appropriate value or by a variable symbol in your application.
- Keywords, which may be used for some message parameters, are specified in uppercase and appear exactly as they should in parameter lists. For example, the Add Item (D009) command of the Dialog Director uses the keyword `TEXT` to add an editable text field.

△ **Important** Text set off this way gives important information that you should read carefully. △

The spaces in the examples are optional and are not significant. However, you may want to avoid using extra spaces, since they increase message traffic.

The examples in this guide illustrate only a few of the many possible ways to write an application to work with MWS.

## What you need

To use the MacWorkStation application, you will need one or two Macintosh computers running System file version 6.0.2 and Finder™ 6.1, or later versions. MWS will run on the Macintosh Plus and later members of the Macintosh family of computers—currently the Macintosh Plus, Macintosh SE, and Macintosh II computers—and requires only a single 800K disk drive.

- ◆ *Note:* Some features of MWS are available only if you are using System file 6.0, or later versions, including text styles and pop-up menus. Also, RGB color and dialog box text styles are available only if you are using a Macintosh computer with Color QuickDraw™ software. Currently, the Macintosh II, Macintosh IIx, Macintosh IICx, or Macintosh SE30 have Color QuickDraw.

If you are using two computers, they must have the proper physical connections in order to communicate with one another. MWS and your application must use the same communication protocol so that messages and data can be sent between them correctly.

---

## The MacWorkStation Developer's Kit

The MacWorkStation Developer's Kit consists of these items.

■ ***MacWorkStation Program Disk***

- ☐ MacWorkStation—the application program
- ☐ Bear Cal—a sample MWS document for use with MWS and the TestHost program
- ☐ Apple Exec—another sample MWS document

■ ***MacWorkStation Samples Disk***

- ☐ TestHost—the MacWorkStation prototyping program
- ☐ Bear Cal script—a document containing a script that runs with TestHost
- ☐ Apple Exec script—another sample script
- ☐ TestHost folder

■ ***MacWorkStation Programmer's Guide*** (this manual)

■ ***MacWorkStation Programmer's Reference***

■ ***Human Interface Guidelines: The Apple Desktop Interface***

The MacWorkStation Developer's Kit includes two Macintosh disks. You should immediately make copies of both disks and save them as backups. One disk, labeled *MacWorkStation Program Disk*, contains the application and several MacWorkStation documents. The documents are examples for use with the MacWorkStation TestHost application, which is explained in the *MacWorkStation Programmer's Reference*.

The second disk, labeled *MacWorkStation Samples Disk*, contains TestHost, an application that lets you learn more about MWS. (This disk was called the *MacWorkStation TestHost Program Disk* for MacWorkStation 3.0.) It also contains several TestHost script documents that work with the MacWorkStation document used with TestHost. The TestHost application lets you make prototypes of your MacWorkStation application, and test your code and resources. To use TestHost, you will need two Macintosh computers connected by a serial line or by LocalTalk™ cables, or a single Macintosh computer running MultiFinder™ system software.

## Development system documentation

The *Apple Technical Library*, published by Addison-Wesley, is a set of technical books from Apple Computer, Inc. that explain the hardware and software of the Macintosh family of computers:

- *Human Interface Guidelines: The Apple Desktop Interface*. This book describes the Apple user interface for the benefit of people who want to develop applications.
- *Inside Macintosh*, Volumes I, II, and III. These books cover the Macintosh User Interface Toolbox and Operating System for the original 64K Macintosh ROM, along with user interface guidelines and hardware information.
- *Inside Macintosh*, Volume IV. This book provides additional information about the Macintosh Plus and Macintosh 512K enhanced computers.
- *Inside Macintosh*, Volume V. This book provides additional information about the Macintosh SE and Macintosh II computers.
- *Inside Macintosh X-Ref*. This reference contains comprehensive indexes, routine lists, and a glossary for *Inside Macintosh* and other Macintosh programming books.
- *Programmer's Introduction to the Macintosh Family*. This book provides an overview of software development for the Macintosh family of computers. It focuses on the differences between event-driven programming and more traditional programming techniques. It covers such topics as QuickDraw graphics, screen displays, and the Macintosh User Interface Toolbox.
- *Technical Introduction to the Macintosh Family*. This book provides an introduction to the hardware and software design of the Macintosh family and serves as a starting point for the *Apple Technical Library*. It is oriented primarily toward the Macintosh Plus, Macintosh SE, and Macintosh II computers, but it also touches on differences in the earlier versions of the Macintosh computer.

Other books that may be helpful include the following, which are available from the APDA™ group (the Apple Programmers and Developers Association).

- *Macintosh Programmer's Workshop Reference*. This guide covers the Macintosh Programmer's Workshop (MPW™) Shell and utilities, including the resource editor (ResEdit™), resource compiler (Rez), linker, Make facility, and debugger.
- *MPW Assembler Reference*. This reference tells you how to prepare source files to be assembled by the Macintosh Programmer's Workshop Assembler.
- *MPW Pascal Reference*. This manual provides information about the MPW Pascal language and the use of the MPW Pascal programming system.
- *MPW C Reference*. This manual tells you how to write C programs that you can link with programs written in MPW Pascal.

## APPLE CONFIDENTIAL

APDA provides a wide range of Apple and third-party technical products and documentation for programmers and developers who work on Apple equipment. Additional copies of this guide and the *MacWorkStation Programmer's Reference* can also be obtained through APDA. For information about APDA, contact

Apple Programmers and Developers Association

Apple Computer, Inc.

20525 Mariani Avenue, Mailstop 33G

Cupertino, California 95014-6299

1-800-282-APDA (1-800-282-2732)

FAX: 408-562-3971

Telex: 171-576

AppleLink: DEV.CHANNELS

If you plan to develop hardware or software products for sale through retail channels, you can get valuable support from Apple Developer Programs. Write to

Apple Developer Programs

Apple Computer, Inc.

20525 Mariani Avenue, Mailstop 51W

Cupertino, California 95014-6299

**△ Important** This manual does not address the issues of licensing the MacWorkStation application. To use MacWorkStation messages or to copy and/or distribute the MacWorkStation program, you must have a valid MacWorkStation License Agreement, which is available from Apple Computer, Inc. △

## Chapter 1 **What Is MacWorkStation?**

THE MACWORKSTATION™ APPLICATION runs on a Macintosh® computer and provides an environment for the Macintosh user to interact with another application, usually running on another computer. This chapter describes the relationship between the MacWorkStation application and this other application, and explains the role of MWS. This chapter also discusses the design principles of Macintosh applications. ■

---

## The MacWorkStation-client system

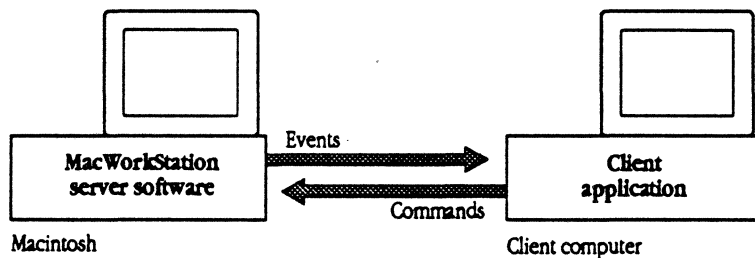
The MacWorkStation application runs on a Macintosh computer, and makes it possible for another application to **manage** the Macintosh user interface presented to the MWS user. This other application that works with the MacWorkStation application is called the **client application**. The MWS user interacts with the client application through the Macintosh user interface.

Usually the client application runs on a computer other than the Macintosh computer. This other computer, called the **client computer**, can be a mainframe computer (such as those made by IBM or DEC) or a personal computer, including another Macintosh computer. In fact, the client application and MWS can run on the same Macintosh computer, using MultiFinder™. The Macintosh computer using MWS is the **server computer**.

This guide refers to all the elements that make-up a single implementation of MWS as the *MacWorkStation-client system*. The MacWorkStation document used with a particular client application is a major element in a MacWorkStation-client system. See "The MacWorkStation Document" in Chapter 2.

The MacWorkStation application and the client application communicate through **messages**. Messages sent from the client application to MWS are called **commands**. Messages sent from MWS to the client application are called **events**. Figure 1-1 shows how these messages pass between MWS and the client application.

■ **Figure 1-1** MacWorkStation message protocol





---

## What MacWorkStation does

The MacWorkStation application provides two main functions. First, MWS provides the environment in which the Macintosh user will work when interacting with the client application. MWS does tasks such as opening windows, saving and opening files, and displaying the standard Macintosh menus. Typically, the user will start a session by opening an MWS document that has been set up previously by the client application programmer. This document contains any necessary components for the application. It usually includes a **Communication Command Language** (CCL) log-on script to establish communications between the server computer and the client computer. It may also include resources used by the client application, such as dialog box or menu bar descriptions.

Second, MWS provides the set of messages that MWS and the client application use to communicate with one another. For example, the client application might send messages to MWS, instructing it to display a particular dialog box. When the user makes a selection or closes the dialog box, MWS sends a message to the client application, reporting the user's choice. MWS also provides communication-level support through **communication modules**, and extensibility through **executable code modules**.

As you create a MacWorkStation-client system, you will notice the following features:

- MacWorkStation messages let developers use the Macintosh interface without having to program on the Macintosh computer. Using MWS, you can develop a variety of applications on different computers, each of which provides a standard human interface on the Macintosh family of computers.
- From a user's standpoint, the MacWorkStation application gives a familiar human interface to information processing provided by the client application.
- The MacWorkStation application will take care of printing, saving files, editing text, and other tasks for the client application.
- MacWorkStation communication modules allow client applications running in different environments and using different communications media, such as standard serial, AppleTalk®, and Ethernet protocols, to use Macintosh display, printing, and file-handling utilities.

---

## Types of MacWorkStation-client systems

A MacWorkStation-client system can be either "generic" or "tightly coupled." In a *generic* system, all the data for such things as menus, graphics, prompts, and windows come directly from the client application. In this system, MWS waits for an action by the MWS user and then transmits the appropriate data back to the client application. It is up to the client application to decide what response is required, if any, at all times. To implement generic systems, programmers need to know only the features of the Macintosh user interface (menus, dialog boxes, and windows) as implemented in MacWorkStation messages. Programmers do not need to know the details of Macintosh program development.

In a *tightly coupled* system, the client application and MacWorkStation components are designed to work together closely. The client application relies more on MWS to handle user actions. Also, the MWS document used with the client application contains user interface resources, scripts for logging on to the client application, executable code modules, and other elements needed by the system. Examples of such applications are a "universal" database query interface and a graphics-based financial modeling system.

In either the generic or the tightly coupled model, the following services are provided by the MacWorkStation application:

- Text editing—All standard text-editing operations, such as text entry, cutting, copying, and pasting, are available.
- Menu management—The standard Apple, File, Edit, Font, and Style menus are provided with MWS. The client application can determine which of these menus to display, but from then on MWS controls them in a totally "local" fashion. Their actions have no effect on the client application.
- Window management—The MacWorkStation user can change window size, scroll windows, send windows to the back, select a window, and move windows without assistance from the client application.
- File access—The MacWorkStation user can save the contents of a window to a disk file and retrieve the contents of a disk file without directly involving the client application.

---

## Why a Macintosh interface?

There are many reasons why you'd want to use the Macintosh desktop interface for your applications rather than the typical interface of a "dumb" terminal. Perhaps the most important one is that the Macintosh computer is easy to use. From the moment users first see the welcoming message, they feel comfortable with Macintosh applications and feel a sense of control over them. The MWS applications you write can provide users with the interface they have come to expect. Users can directly manipulate documents they are working on. They see on the screen what they are doing, and they can select actions from alternatives presented on the screen.

---

## Human interface guidelines

The ease of use that users expect from the Macintosh computer is the result of close attention paid to the many ways people use computers. A study of users' actions has resulted in a set of ergonomic principles, detailed in a publication called *Human Interface Guidelines: The Apple Desktop Interface*. You should study these guidelines, which are part of the MacWorkStation package, before you begin programming with MWS.

As explained in *Human Interface Guidelines*, a human interface is more than a visual display. In fact, it is possible to have a human interface without any visual display. A human interface is the sum of all communication between the computer system and the user. The human interface presents information to the user and accepts information from the user. It is the way in which the user accesses the power and information of the computer.

Among the points expressed in these guidelines is the idea that Macintosh applications should be easy to learn and to use. Applications should build on skills people already have, not require them to learn new ones unnecessarily. The user should feel in control of the computer, not the other way around. MacWorkStation messages let you make your applications attractive and easy to use, as recommended in *Human Interface Guidelines*. As you read the sections that follow, refer to these guidelines, if you have questions about how standard Macintosh features should work.

### **Three qualities of a good program**

If you are familiar with programs written for the Macintosh computer, you know that they have certain characteristics that give them a "feeling" of belonging on a Macintosh computer. It is this "feeling," provided by the user interface, that you will want to incorporate into your programs.

A good Macintosh application embodies three qualities: responsiveness, permissiveness, and consistency.

#### **Responsiveness**

With a responsive application, a user's actions tend to have direct results. If a user chooses an italic font, the words on the screen change to italic right away. If a user selects an icon, the icon is highlighted immediately. With pull-down menus, the user can choose a command directly and instantaneously.

#### **Permissiveness**

Users make mistakes. They also explore the application in order to learn to use it. If your application is permissive, it allows users to make mistakes and to explore without penalizing them for doing something wrong. Your application should let users accomplish their tasks spontaneously and intuitively. If the user makes a mistake, your application should present a clear and instructive message about the problem. However, these messages should occur in your program only when necessary.

#### **Consistency**

All applications should be consistent, so that a user moving between applications does not need to learn a new interface. Consistency is easy to achieve for a MacWorkStation-client system because the MacWorkStation messages call the same routines used to implement the Macintosh user interface in other applications.

---

## General suggestions for application programs

Here are some suggestions to follow when developing a MacWorkStation application that will make your application more user-friendly. A more complete list of suggestions can be found in the *Human Interface Guidelines*.

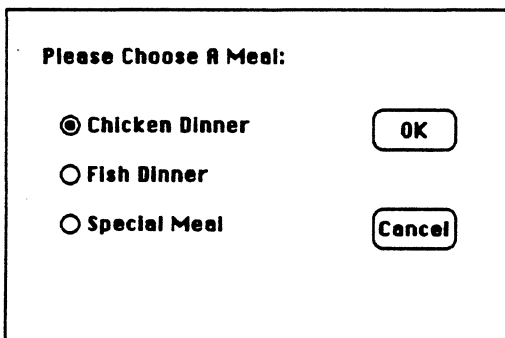
---

### Modes

A **mode** is a part of an application that the user has to formally enter and leave. Modes restrict the operations that can be performed while they are in effect. Modes can vary from picking a different-sized paintbrush in a graphics editor to holding down the Shift key. You do not need to completely avoid using modes. However, you should use them sparingly, because using them too often will give users the feeling that the application is unnecessarily restrictive and unfriendly.

Your application should always give a clear, visual indication of the current mode. The visual indication that you display should be near the object most affected by the mode. You should make it easy to get into or out of the mode. However, entering and leaving the mode should be the direct result of the user's actions. For example, a modal dialog box has a standard appearance in Macintosh applications. Figure 1-2 shows an example. A modal dialog box informs the user of the result of some action, and requires the user to explicitly dismiss it (by clicking the OK button, for example) before doing anything else.

■ Figure 1-2 A modal dialog box



## Graphics and color

You can take advantage of the high-resolution Macintosh screen by creatively using graphics in your applications, even in places where other applications might use text. As much as possible, commands, features, and parameters of an application should appear as graphic objects on the screen that suggest the use of the object. In the Macintosh user interface, objects such as the trash can icon resemble the familiar objects whose functions they emulate. Objects that act like push buttons will "light up" when pressed.

Dialog boxes and menus are other examples of the use of graphics that are familiar to Macintosh users. You will probably want to use them in your applications.

MacWorkStation messages support the color capabilities of **RGB monitors**, so you may wish to use color in your applications. Be sure to consider that some people have problems distinguishing colors and that not every user may have an RGB monitor. Generally, you should not base the ability to use your software on a user's ability to identify colors, although colors can enhance the overall look of your applications.

## Chapter 2 Using MacWorkStation

THE MACWORKSTATION-CLIENT SYSTEM consists of several elements that work together to provide a service to a Macintosh user. This chapter explains what these elements are and what you need to do to create the MacWorkStation-client system. Some of the terms used in this chapter may be unfamiliar to you. They will be explained in detail later in this guide. For now, just read along to get the general idea. ■

---

## Creating the MacWorkStation-client system

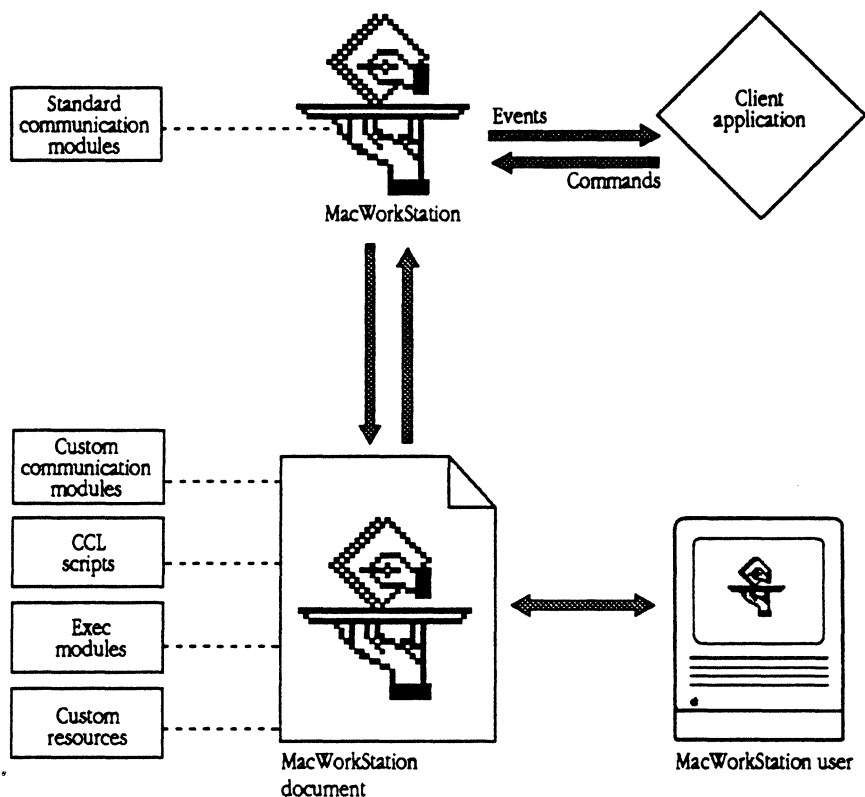
The MacWorkStation-client system consists of a client application, a Macintosh computer running the MacWorkStation application, an MWS document with a CCL script, and a communication module available in the MacWorkStation application or the MWS document. Typically, the client application will be running on another computer, so you will need a hardware connection between the computers. Your system may also use custom alerts, cursors, dialog boxes, graphics, menus, windows, and executable code modules, which can be stored as resources in the MWS document. Here are the things you need to do to set up a MacWorkStation-client system:

- Write a client application that will tell MWS what to do.
- Create a MacWorkStation document to use with your system.
- Create a Communication Command Language (CCL) script in the MacWorkStation document that will establish the communication link between your client application and MWS.
- Be sure the correct communication module for your application is stored as a resource either in the MacWorkStation application or the MWS document. The communication module manages the communication protocol. The **CCL script** must call this module during the log-on procedure. Communication modules for two serial protocols and AppleTalk are provided with MWS. You can also use communication modules that you create. See the *MacWorkStation Programmer's Reference* for more information about communication modules.
- Create executable code modules for any modifications you need to make to MWS, and store their code resources in the MWS document. See the *MacWorkStation Programmer's Reference* for more information about executable code modules.
- Create any user interface resources your system needs, such as custom dialog boxes or menus, and store these resources in the MWS document. See the *MacWorkStation Programmer's Reference* for more information about using resources.



Figure 2-1 shows the relationship of these elements in the MacWorkStation-client system.

■ **Figure 2-1** The MacWorkStation-client system



## Starting a MacWorkStation-client system

When you have prepared the various elements for your MacWorkStation-client system, you need to do these things to start up the system.

- Set up the hardware connection between the computers.
- Start your client application. The client application needs to be ready to receive and process a message from the MacWorkStation application in order to start the session.
- Open your MacWorkStation document. You can open the MWS document from the Finder™ or use Open from the MWS File menu.
- If you have never used the document to **log on** to the client application, you will be asked to select the CCL script to use. MWS then attempts to log on to the client application, using the

CCL script you select and the communication module that the script indicates. (If the CCL script logs on successfully, MWS will use this script in the future to log on to the client application whenever the document is opened.)

When the script logs on to the client application, MWS sends a message to the client application and waits for a reply. After replying, the client application can begin the session, interacting with MWS and the user as the session continues.

When the session is finished, the CCL script should do what's necessary to log off from the client application. Typically, the session is ended when the user chooses Disconnect from the Apple menu, or Close or Quit from the File menu.

---

## The client application

You can write your client application in any programming language that works with the client computer. During initial development, you may wish to use TestHost with MWS. TestHost is an application that allows you to emulate your client application by sending commands to MWS and responding to events sent from MWS. Since TestHost also runs on a Macintosh computer, you'll find it easy to use to design your application's user interface, and to test and debug your program. You can also use TestHost to create dialog boxes and save them as resources in your MWS document. TestHost is explained in the *MacWorkStation Programmer's Reference*.

You may also use MWS Event Handler to create a prototype of your application. You can use MWS Dialog Builder to create and edit dialog boxes you use with your client application. MWS Event Handler and MWS Dialog Builder are available from APDA™. (Information on how to contact APDA appears in the Preface.)

The client application must be running in order to receive an event sent by MWS at the start of each session. This message is called the MacWorkStation Online (P256) event, which MWS sends after the log on is successful. The client application must then send the Host Online (P001) command to MWS in order for the session to begin.

After this exchange of messages, your client application can be in complete control. However, you can let the MacWorkStation application take care of many tasks, which saves you development time and reduces communications traffic. These tasks include opening, closing, and saving files; redrawing windows; printing; supporting text editing; and many others.

You probably want your application to respond to the user's actions in the flexible manner familiar to Macintosh users. You can achieve this Macintosh-type behavior by using event loop programming techniques described next.

## The event loop

The event loop is the central routine of any Macintosh application that supports the Macintosh user interface. Using the event loop, an application doesn't expect events to occur in a particular order. Instead, it constantly checks for inputs, such as mouse actions and keystrokes, that can occur in any order. The client application can then respond to any event in an appropriate way.

This approach to programming contrasts with programs that require the user to make requests or perform tasks in a specific order. Instead, the emphasis is on responding to any request the user makes at any time. This approach enables the widest possible range of user activities. For example, there's no reason not to let the user set printing options before there's anything to print.

Using the event loop, a client application can easily handle events generated in response to commands it sends, by simply waiting for MWS to return a response. The client application can also use the event loop to process events initiated by the user, by first identifying the event, then responding to it.

The basic structure of an event loop is very simple. Here is an example of the main routine of a client application using event loop programming.

## APPLE CONFIDENTIAL

```
main()
{
    char    EvtClass;
    int     EvtID;
    char    EvtParms[256];

    AllInit();
    do
    {
        GetMWS(&EvtClass, &EvtID, EvtParms);
        switch(EvtClass)
        {
            case 'A':
                DoAEvent(EvtID, EvtParms);
                break;

            case 'D':
                DoDEvent(EvtID, EvtParms);
                break;

            case 'F':
                DoFEvent(EvtID, EvtParms);
                break;

            case 'G':
                DoGEvent(EvtID, EvtParms);
                break;

            case 'L':
                DoLEvent(EvtID, EvtParms);
                break;

            case 'M':
                DoMEvent(EvtID, EvtParms);
                break;

            case 'P':
                DoPEvent(EvtID, EvtParms);
                break;

            case 'T':
                DoTEvent(EvtID, EvtParms);
                break;

            case 'W':
                DoWEvent(EvtID, EvtParms);
                break;

            default:
                break;
        }
    }
    while (!AllDone);

    AllDispose();
}
```

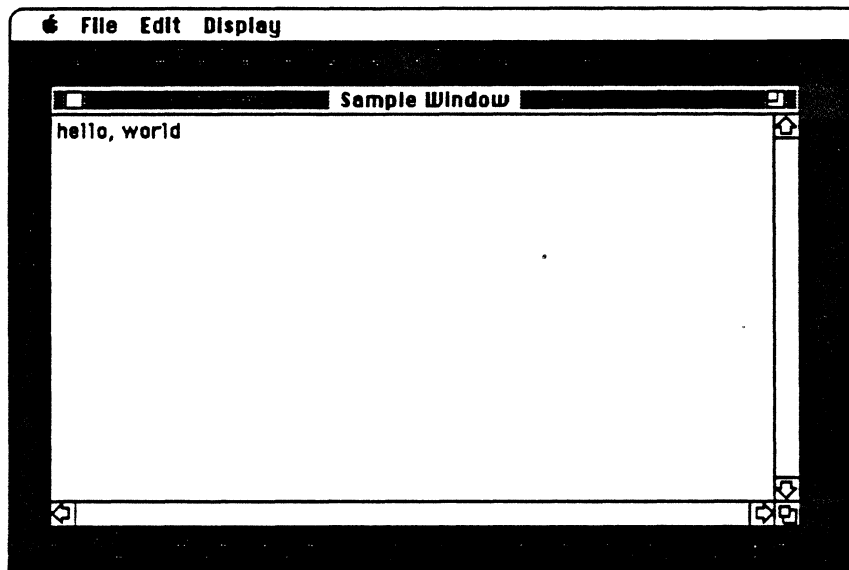
---

## About the "Hello.C" sample application

The example application called "Hello.C" is familiar to most Macintosh programmers. It is a simple Macintosh application program that illustrates the principles of creating the Macintosh user interface. Although it is too simple to be a practical, it shows the overall structure that a client application will have, and it does several of the things that any client application will do.

"Hello.C" creates a single window, "Sample Window," which displays the message "hello, world" (Figure 2-2). The window has a title bar, title, close box, and resize box. The user can resize the window by dragging the resize box, and can move the window around the desktop by dragging it by its title bar. The application displays four menus: the standard Apple menu, from which the user can choose desk accessories, and File, Edit, and Display. Using the File menu, you can open existing text files. If the window is too small to display all the text it contains, the scroll bars become active so you can view the text. You can also print and save files. You can type new text, and select text in order to edit it. Using the Edit menu, you can cut, copy, and paste the text. Clicking the close box closes the window.

■ **Figure 2-2** Screen created by a simple application



## The "Hello.C" sample application

The "Hello.C" client application produces the screen shown in Figure 2-2. Comments are included in the listing so that you can see which groups of MacWorkStation messages are being used and what operation is being performed.

```

/* This is a sample client application which:
    1) displays the standard (Apple, File, and Edit) menus;
    2) puts up a standard text window;
    3) displays the string 'Hello world' in the window.

*** This version runs under VMS or UNIX. It should work with any C. ***
*** However, it may not run under other versions of C unless changes ***
*** are made so that the MWSPut and MWSGet routines address a      ***
*** serial port.                                                    ***

*/

#include <stdio.h>

static char quitMWS = 0;      /* This will be set to 1 in response to a
                               Quit event (P257) from MWS
                               */

main()

/* The main loop of the program simply waits for an event from
MWS, then responds to the event. Note that in this simple
example, the only events recognized are:
    P256 - MacWorkStation Online
    P257 - MacWorkStation Offline
*/
{
    char    evtClass;
    int     evtID;
    char    evtMsg[512];

    printf("%s", "MWS GO\n");    /* Send this so the CCL knows we're running */

    do {
        if (MWSGet(&evtClass, &evtID, evtMsg))
            if (evtClass == 'P')
                doPEvt(evtID, evtMsg);
    } while (! quitMWS);
}

```

## APPLE CONFIDENTIAL

```
doPEvt (evtID, evtMsg)
```

```
/* The doPEvt function responds to Process events from MWS.
   P256 is the first event sent when MWS commences.
   P257 is sent when MWS is quitting.
*/
```

```
int     evtID;
char    *evtMsg;
```

```
{
    switch (evtID) {
        case 256 :
            appStartup();
            break;
        case 257 :
            quitMWS = 1;
            break;
    }
}
```

```
appStartup()
```

```
/* The appStartup function first responds to the P256 event, then
   puts up the menu bar, displays a text window, and sends some
   sample text to the window.
*/
```

```
{
    MWSPut("P001", "");                /* Issue handshaking sequence */
    MWSPut("M004", "0;FILE;EDIT;TEXT;"); /* Install default menus */
    MWSPut("W001", "1;TEXT;Sample Window;;TRUE;TRUE");
                                         /* Create a text window */
    MWSPut("T010", "1;Hello World");    /* Put some text in the window */
}
```

## APPLE CONFIDENTIAL

MWSGet(evtClass, evtID, evtMsg)

```
/* The MWSGet function gets a message from MWS and breaks it into its
   components. The protocol expected is:
       [      message begin character
       (data) message contents
       \n      message end character (simplifies using gets)
   This conforms to the ID=2 transport-layer protocol.
   */
```

```
char    *evtClass;
int      *evtID;
char     *evtMsg;

{
    char    begMsg;
    char    mwsMsg[512];

    gets(mwsMsg);
    sscanf(mwsMsg, "%c%c%3d", &begMsg, evtClass, evtID);

    if (strlen(mwsMsg) > 4)
        strcpy(evtMsg, &mwsMsg[4]);
    else
        *evtMsg = '\0';
    return 1;
}
```

MWSPut(cmdClass, cmdParms)

```
/* The MWSPut function sends a message to MWS. The protocol is:
       [      message begin character
       (data) message contents
       \n      message end character
   This conforms to the ID=2 Serial transport-layer protocol.
   */

char    *cmdClass;
char     *cmdParms;

{
    printf("%c%c%s%c", '[', cmdClass, cmdParms, '\n');
}

/* End of Hello.c */
```



---

## The MacWorkStation document

The MacWorkStation document is the user's access to any MacWorkStation-client system. Users interact with the client application through the MacWorkStation document. Usually a user will need to open a specific MWS document to work with a particular client application. The user opens the document to start the session with the client application.

When you launch the MacWorkStation application, you are asked to choose a document to open, or to create a new one. To create a new MacWorkStation document when MWS is running, you choose New from the File menu. When you create a new MWS document, you are also asked to create a CCL script for the document. The CCL script is used to establish communications between the computers.

The MacWorkStation document will store many of the elements of your MacWorkStation-client system. It will contain the CCL script. It might store a custom communication module used by the script.

The MWS document can also store **resources** you have created to use with your system. Menus, dialog boxes, cursors, windows, and many other features of the Macintosh interface are resources. You can create your own resource whenever a standard MacWorkStation resource does not meet your needs. Several programs are available for creating your own resources. MWS Dialog Builder, ResEdit™, and MPW™ Rez are programs available from APDA. (Information on how to contact APDA appears in the Preface.) Resources are a unique feature of Macintosh applications. Refer to *Inside Macintosh* to learn more about resources.

The MWS document may also include executable code modules that extend the functionality of the MacWorkStation application. See the *MacWorkStation Programmer's Reference* for additional information.

---

## MacWorkStation-client communications

For the MacWorkStation application and the client application to interact, they must establish some form of communication. Besides the actual hardware, two elements are required to make the link: the CCL script, and the communication module. These two elements are stored in MWS or the MWS document. The only requirement of the client application is that it be able to send data over the communication link you have chosen.

You create the CCL script and save it in the MWS document. The CCL script establishes the actual communication link between MWS and the client application when the MWS document is opened. If the CCL script logs on successfully, the session can proceed. The CCL script may also contain instructions for disconnecting at the end of the session. The Communication Command Language provides the necessary commands to set up and control the communication session. The CCL commands and the process of creating a CCL script are explained fully in the *MacWorkStation Programmer's Reference*.

The communication module handles the low-level aspects of the communication session. Apple Computer, Inc. provides a number of communication modules, including two serial protocols and an AppleTalk protocol. If none of these protocols is acceptable, you can develop a custom protocol or use an acceptable protocol from a third-party developer.

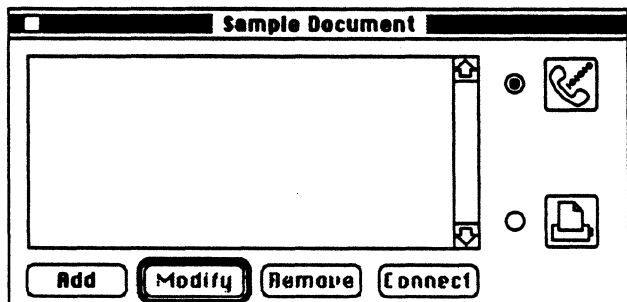
---

### Communication Command Language (CCL) scripts

CCL scripts contain the instructions for logging on and logging off the client application by using the Communication Command Language. CCL scripts make it possible for the user to log on and log off automatically without having to decipher different operating-system prompts or enter commands. The first task of the CCL script is to designate the communication module used in the session. Then, the CCL script waits for prompts from the client computer and responds appropriately to requests for information. The CCL script can also request information from the user, such as a password or identification code.

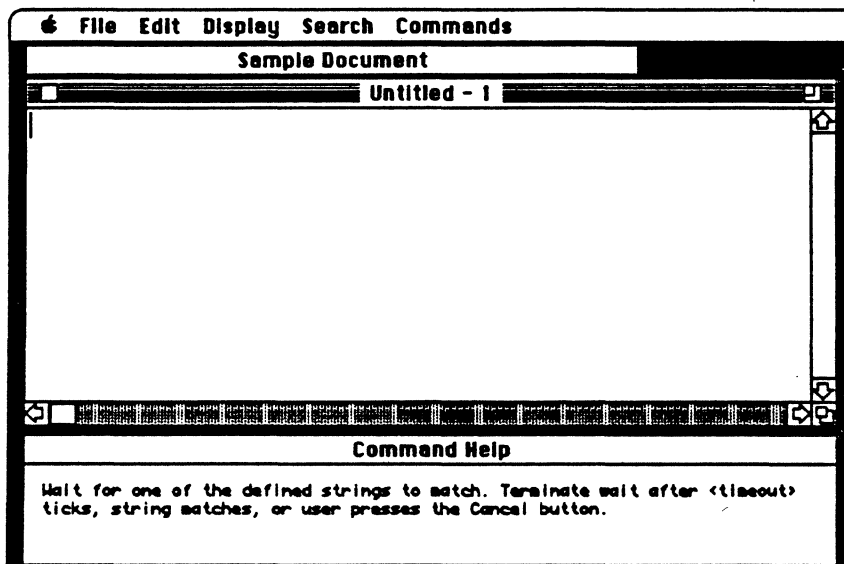
You are asked to create a new CCL script when you create a new MacWorkStation document. After naming the MacWorkStation document, you will see the choose-script dialog box shown in Figure 2-3.

■ Figure 2-3 Choose-script dialog box



When you click the Add button, you'll see a CCL script-edit window, in which you will type the commands to create the CCL script. If you need help, you can choose a command from the Commands menu. A brief description will appear in the Command Help window. Figure 2-4 shows a CCL script-edit window with the Command Help window.

■ **Figure 2-4** Script-edit window with Command Help window



To write a CCL script, you must know exactly what prompts will come from the client application. CCL scripts may also contain commands needed to log off from the client application when the session is completed. The commands in the log-off sequence are preceded by an asterisk (\*). Here is an example of a log-off command:

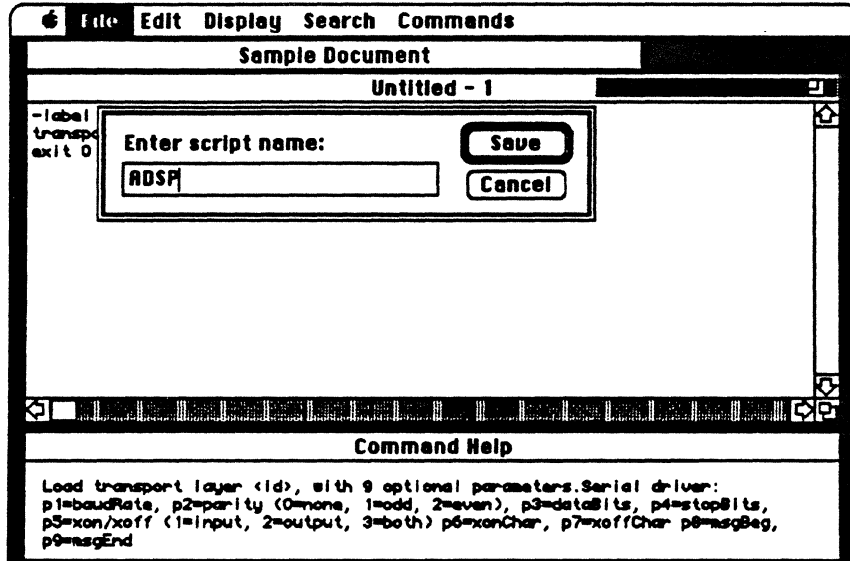
```
*xmit "logout"
```

All the log-off commands must appear together at the beginning of the CCL script, before any commands in the log-on sequence.

## APPLE CONFIDENTIAL

After you type your script, you choose Save from the File menu. The dialog box shown in Figure 2-5 appears so that you can name your CCL script. You type in a name for the script and click the OK button. You have now created a CCL script.

■ Figure 2-5 New script name dialog box



The next time you open the MWS document, you will be prompted to select the CCL script to use to log on to the client application. Once you have logged on successfully with a CCL script, that script is executed whenever the document containing it is opened. You can prevent the automatic execution of the CCL script by holding down the Option key when you open the document. You will see the choose-script dialog box shown in Figure 2-3. You can then add a new CCL script to the document, modify or remove scripts, or select a different script to use to log on.

---

## Communication modules

A communication module manages a communication (transport-layer) protocol for the connection between MWS and the client application. Communication modules are code resources stored in the MWS document and called by the CCL log-on script, using the CCL `TRANSPORT` command.

A wide variety of communication networks can be used by a MacWorkStation-client system. Several communication modules are provided with MWS:

- A serial communication module that uses start and stop message characters.
- A serial communication module that sends a Begin Message character and then a 2-byte count of the number of bytes in the message.
- A communication module that sends and receives messages over the AppleTalk network. AppleTalk is Apple's local area network for connecting Macintosh computers with each other and with other shared resources. If you want to use this communication module, your client application must have access to AppleTalk and be capable of supporting the AppleTalk Data Stream Protocol (ADSP). The ADSP connection includes error checking, a benefit of using AppleTalk as the communication medium.

Other communication modules may be available from APDA. (Information on how to contact APDA appears in the Preface.) You can also use custom communication modules.

See *MacWorkStation Programmer's Reference* for more information about communication modules. Information about creating custom communication modules is available from APDA.



## Chapter 3 Directors and Messages

MACWORKSTATION COMMANDS AND EVENTS are grouped together by function into **directors**. There are eleven directors. This chapter gives an overview of the function of the directors. This chapter also discusses the structure of MacWorkStation messages. Additional information about the MacWorkStation directors can be found in the *MacWorkStation Programmer's Reference*. ■

---

## MacWorkStation directors

Each MacWorkStation director is a group of commands and events involved with a particular aspect of the application. For example, the Cursor Director provides commands to change and control the cursor. Many of the directors' commands interface with routines belonging to the Macintosh Operating System and the User Interface Toolbox **managers**. The managers are groups of functionally related commands available to Macintosh applications. See *Inside Macintosh* for more information about the Macintosh Operating System, the User Interface Toolbox, and the toolbox managers.

Here is a brief description of the MacWorkStation directors. A fuller explanation of the directors, with examples, can be found in Chapter 4. The commands and events for each director are described in the *MacWorkStation Programmer's Reference*.

- **Alert Director**—Allows a client application to display notes and warning messages to the user. Alerts can range from “stop alerts,” which warn the user that some problem is about to arise, to “note alerts” that simply require users to acknowledge the alert by clicking the OK button. The client application can use the standard alerts, including a speaker beep, stop alert, note alert, caution alert, message banner, and picture banner. The client application can also use custom alerts, which might include color, that are saved as resources in the MWS document.
- **Cursor Director**—Makes it possible for a client application to hide or change the form of the pointer visible to the MWS user. Your application can use the standard cursors, including the arrow, wristwatch, and I-beam. Custom cursors, including color cursors, may also be displayed.
- **Dialog Director**—Lets a client application present Macintosh dialog boxes. The client application specifies what each dialog box should look like and what it should do. A dialog box can have mixed text fonts, sizes, and styles, as well as horizontal and vertical scroll bars, and size and zoom boxes. Dialog boxes can include text input fields, different types of buttons, and pop-up menus. Dialog boxes or items within them may be displayed in color. Dialog boxes can be saved as resources in the MWS document for use with the application. The client application calls each of these dialog boxes by using its unique resource ID.
- **File Director**—Lets a client application manipulate Macintosh files. The client application can create, delete, and rename files, get and set file information, and open and close files. Macintosh files consist of a resource fork and a data fork, and independent transfer of both file forks is supported. File information may include specific access rights when opening the file, and byte ranges to be locked within a file. The application can set directory access privileges or check a directory's access privileges.
- **Graphics Director**—Provides access to many of the QuickDraw™ graphics routines that are built into the User Interface Toolbox. These routines allow the client application to draw lines, geometric shapes, patterns, and text. The application can draw graphic items in color. QuickDraw pictures can be transferred between the client application and Macintosh windows. The contents of graphics windows can be saved as MacDraw PICT documents.



- **List Director**—Lets the client application present information in a list format or in a format similar to a **spreadsheet**. List windows can be saved as Microsoft Excel TEXT documents. Records and fields can be transferred between the client application and Macintosh windows. Individual fields may be edited. Records can be sorted in ascending or descending order, based on one or more fields, and applications can be set up to read only those records modified since the last reading.
- **Menu Director**—Lets the client application choose from several standard menus such as File, Edit, Font, and Size. MWS handles any actions caused by the user's choice of items from these menus. The client application can use the Menu Director to create custom menus, including hierarchical and color menus. The Menu Director is also used to create pop-up menus, which can then be used in dialog boxes. The client application can use several menu bars, each tied to a different document window on the Macintosh computer.
- **Process Director**—Handles certain administrative tasks, such as placing the Macintosh computer in a wait state while the client application performs other tasks.
- **Text Director**—Allows the client application to create and manipulate text in text windows. Text windows may contain mixed fonts, sizes, and styles, as well as color text. The client application or user can display and edit the text, and save it into MacWrite TEXT documents. Text can be transferred between the client application and Macintosh windows.
- **Window Director**—Lets the client application create windows of various sizes, shapes, colors, and functions. Your application can check for a user action on a window such as activate, deactivate, move, or resize. Other directors, such as the Text, List, or Graphics Director, provide the window's contents. Windows can represent editing areas, display areas, or entire Macintosh documents.
- **Exec Director**—Lets you use your own executable code modules (exec modules). You use exec modules to extend and modify MWS. The Exec Director includes commands that initialize, terminate, and control executable code functions. The structure of executable code is explained in the *MacWorkStation Programmer's Reference*.

## MacWorkStation messages

At the heart of the MacWorkStation-client system are the messages exchanged between the client application and MWS. These messages are either commands or events. A command is a message sent from the client application to the MacWorkStation application, instructing it to perform some action, such as creating a window.

An event is a message sent from MWS to the client application. Events are sent under two conditions. First, some events are sent in response to commands from the client application. For example, a client application might send `F013 1;` (the Get File Fork Size command). MWS might respond with `F261 1;5432;` (the Get File Fork Size Response event). This event tells the client application that the size of the requested file is 5432 bytes.

Second, some events are sent in response to user actions. For example, if a user chooses a menu item, MWS will send `M256 0;1;2;` (the Menu Selection Event). It tells the client application that the user selected item number 2 from menu number 1 in menu bar 0.

Here is another example. The dialog box in Figure 3-1 is from the Bear Cal sample application. If a user clicks the OK button in a dialog box, MWS will send `D257 1;T;` (the Control Item Pressed event). This event is generated whenever the user clicks a button (such as OK, Cancel, or one of the credit card buttons) in the dialog box.

■ **Figure 3-1** Dialog box from the Bear Cal sample application

Bear Cal Reservation System			
Passenger:	C. Hutson		OK
Destination:			Cancel
Depart:		Return:	
Airline:			
Time:			
Gate:			
Incidentals:	Class:	Seating:	Payment:
<input type="checkbox"/> Smoking	<input type="radio"/> First	<input type="radio"/> Window	
<input type="checkbox"/> Dinner	<input checked="" type="radio"/> Coach	<input checked="" type="radio"/> Center	
<input type="checkbox"/> Movie	<input type="radio"/> Business	<input type="radio"/> Aisle	
<input type="checkbox"/> Rental Car			

The next sections describe the parts of a MacWorkStation message in more detail.

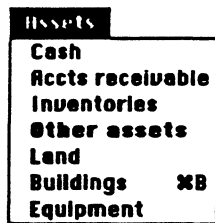
## The structure of a message

A MacWorkStation **message** is a stream of bytes consisting of three parts: a **class**, an **identifier** (ID), and **parameters**, if any. The message class indicates the MWS director involved in the message. The ID **designates** a particular command or event. Finally, many of the commands and events send additional information in parameters. How the messages are formed and sent between the computers is an issue for lower-level communications software.

Here is an example of a command message where the class is **M**, the identifier is **004**, and there are several parameters. This is a Menu Director message that creates the Assets menu, shown in Figure 3-2.

```
M004 0;Assets,20,Cash,Accts receivable,Inventories,Other assets<B,
Land,Buildings/B,Equipment;
```

■ Figure 3-2 Macintosh menu example



---

## Class

The class is a single uppercase ASCII character that determines which MacWorkStation director the message refers to. Table 3-1 gives the MacWorkStation classes and their respective MacWorkStation directors.

■ **Table 3-1** Classes and directors

Class	Director
A	Alert Director
C	Cursor Director
D	Dialog Director
F	File Director
G	Graphics Director
L	List Director
M	Menu Director
P	Process Director
T	Text Director
W	Window Director
X	Exec Director

---

## Identifier

The identifier specifies the particular command or event of the director. It is composed of three ASCII digits, such as 123. All three digits must be given, so an identifier of 1 would be represented as 001. Command identifiers range from 001 to 255. Event identifiers range from 256 to 511.

---

## Parameters

A list of parameters may follow the class and identifier. These parameters provide additional information for the execution of the command or event. Parameters may be required or optional. Some parameters can be repeated a number of times in the same message. For example, to create menus, the client application must indicate the names and items for the menus. These names and items would appear as parameters in the command message.

Parameters may be different data types. Parameters can be **strings**, **integers**, **Boolean operators**, or binary data. With the exception of binary data, the bytes that make up a MacWorkStation command or event are displayable, extended 8-bit ASCII characters. For instance, the numeric value 42 will appear in a parameter list as 42, not as a binary value.

Some parameters are combinations or variations of the basic types. These parameters are indicated in the **examples** in this guide, and in the *MacWorkStation Programmer's Reference*, by using special **parameter** types and keywords.

Because **parameters** can have a number of components, delimiters are used to separate them. Semicolons **delimit parameters**. Commas delimit the components of compound parameters. For instance, the item list of a menu is delimited by semicolons, but the component strings of the item list are separated by commas.

MacWorkStation messages use these parameter types:

- **String**—A sequence of characters whose length should not exceed 255 characters. A string may be enclosed by single or double quotation marks. A string must be enclosed in quotation marks if it contains a semicolon or comma.
- **Integer**—A sequence of characters, each of which has a value in the range 0–9, with an optional leading minus sign (–). An integer value must be within the range that can be specified by a 32-bit binary value.
- **Flag**—Boolean values that are either true or false. The value of a flag parameter may be specified as a string with a first character that is either *T* or *t* for true, or *F* or *f* for false.
- **Bytes**—An arbitrary string of 8-bit bytes delimited only by the end of the message. A parameter of this type may vary in length up to the maximum message size (512 bytes), and may contain any characters (including semicolons or other delimiters).
- **Color**—Some commands use the eight original QuickDraw color keywords: BLACK, BLUE, CYAN, GREEN, MAGENTA, RED, WHITE, and YELLOW. If Color QuickDraw is available, some commands can set RGB color as a compound parameter. RGB color is denoted by three integers in the range of 0 to 65535, separated by commas.
- **Compound**—A parameter that consists of a combination of parameter types.
- **Directory**—A parameter that specifies the name of a volume, folder, file, or the pathname for a folder or file.
- **Keyword**—A string-type parameter with a fixed set of values that your application may send. Keywords are specified exactly as they should appear in the parameter list and are always in uppercase. For example, when creating menus, you may use a keyword to indicate a standard MacWorkStation menu. Standard menu keywords include FILE, EDIT, and FONT.
- **Point**—Two integer values that specify a coordinate. They appear in the parameter list as "*h, v*" (horizontal and vertical coordinates). Notice that the integers are separated by a comma.
- **Rect**—Four integer values that specify a rectangle. They appear in the parameter list as "*left, top, right, bottom*". Notice that the integers are separated by commas.
- **Signature**—A string of exactly four characters, usually used to indicate the type or creator of a file. A signature is case sensitive, so the value 'text' will not be the same as 'TEXT'.

---

## Using aliases

Each **object** created by the MacWorkStation-client system, whether it is a window, dialog box, or open file fork (resource or data), is assigned an **alias** when it is created. The object's alias is one of the parameters used in the commands and events that control the object. The client application and MWS both refer to the object by its alias. If the object is created by the client application, then the programmer specifies the alias to use. For example, if the client application wishes to create a window, it must supply a unique alias that the client application and MWS can use to refer to that window in subsequent messages. In this case, the alias is an arbitrary number. Objects are sometimes created by the MacWorkStation application independent of the client application. In this case, MWS will provide the unique alias for the object it generates. For instance, if the user chooses New from the standard File menu, MWS will create a new text window.

---

## Examples

Here are some examples of commands and events, with an explanation of their features.

W001 5; TEXT; Daily News

This is a window command, so the class is **w**; the identifier is **001**, which falls within the range of a command (1-255). This command will cause the Macintosh computer to put up a standard document window. There are three parameters, separated by semicolon delimiters. The window has an alias of 5. This alias was assigned by the application programmer and will be used subsequently to refer to the window. The keyword **TEXT** specifies a window with a text view and scroll bars. The title of the window is Daily News.

M004 0; FILE; EDIT; Mail, 6, Send, Receive, Address Book

This is a menu command: The class is **M**; the identifier is **004**, which falls within the range of a command. This command will cause MWS to add three menus to the menu bar: File, Edit, and Mail. There are four parameters, separated by semicolon delimiters. The first parameter, 0, is an alias referring to the default menu bar where the menus will be added. The second and third parameters are keywords that specify the File and Edit menus, which are standard menus managed by MWS. The fourth parameter is a compound type consisting of several subparameters separated by commas. The Mail menu has an alias of 6 and three items: Send, Receive, and Address Book.

M256 0; 6; 3

This is a menu event: The class is **M**; the identifier is **256**, which falls within the range of an event. There are three **parameters**, separated by semicolons. In this case, MWS is informing the client application that the user has chosen menu bar 0, menu 6, item 3. This choice corresponds to the Address Book menu item from the previous example.

## Chapter 4 Using Directors

MACWORKSTATION DIRECTORS provide programmers with the tools necessary to create their MacWorkStation-client systems. This chapter uses examples from a simple client application to explain the use of the director commands and events. The *MacWorkStation Programmer's Reference* gives a complete description of all the commands and events used in the examples. ■

---

## About the "Bear Cal" application

"Bear Cal" is a simple example of a client application. The application is designed to be used by reservation clerks of a fictional airline, called Bear Cal, to schedule reservations and perform other tasks. You will find the source code for "Bear Cal" in the appendix, as well as on the *MacWorkStation Samples Disk*. "Bear Cal" is written in MPW 3.0 C.

After initialization, the application enters an event loop to wait for the first action to process. The event loop begins with the `do` command and ends with the `while(!AllDone)` command. The event loop first processes messages with the `MWSGet()` routine. This routine identifies the director class of the message, which is then used in the event loop to determine the appropriate routine to handle the event.

The first event received from MWS will be the MacWorkStation Online (P256) event. When "Bear Cal" receives this event, it executes a startup routine that first responds with the Host Online (P001) command. "Bear Cal" then creates menus, several windows, and a dialog box. When it has finished, the application displays the Flight Schedule window and again waits for the reservation clerk's next action.

The reservation clerk's actions may include scheduling a new reservation or changing an existing reservation. The reservation clerk can indicate the method of payment, and receive an authorization code for some methods. The reservation clerk can set up meals for a reservation, including instructions for special meals, and note whether the customer wants to rent a car.

Other routines of the "Bear Cal" application handle such tasks as determining which menu item a reservation clerk has chosen, or what items have been selected or entered in a dialog box. Finally, when the reservation clerk chooses Disconnect from the Apple menu, or Close or Quit from the File menu, "Bear Cal" processes the event sent by MWS and discontinues the session.

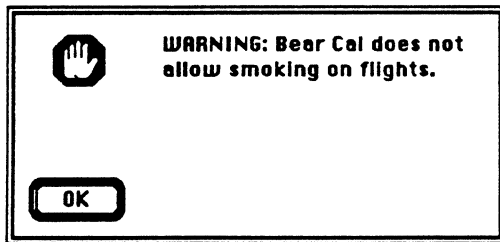


## The Alert Director

The Alert Director commands make it possible for your application to inform users of error conditions or other situations that require their attention. An alert can be a simple beep from the Macintosh speaker or an **alert box** asking the user whether to continue or cancel an action. You can use the standard alerts provided by MWS, or you may choose to create your own.

An alert places the MacWorkStation application in a modal state. The user must respond to the alert before the application can continue. *Inside Macintosh* explains the use of alerts in detail. Figure 4-1 shows an example of an alert.

■ **Figure 4-1** Alert created with the Alert Director



The following command created the alert displayed in Figure 4-1:

```
MWSPut("A002","F;WARNING: BearCal does not allow smoking on flights.");
```

- **A** is the class for the Alert Director.
- **002** is the identifier for the command Stop Alert.
- **F** is the flag for the *respond* parameter. In this case it is false, indicating that no event will be returned to the client application when the user dismisses the alert.
- The final parameter is the *message* that will be displayed in the alert box.

The alert created by this command contains the standard note icon, an OK button, and the text of the message.






## The Cursor Director

The Cursor Director makes it possible for the client application to change and control the mouse pointer on the Macintosh screen. Generally, it is best to let the MacWorkStation application control the pointer, which is the default situation. However, the client application can determine the appearance of the pointer, hide the pointer, or show the pointer.

The pointer can relay information to the user about the state of the application or the options available, as well as reassuring the user about what's happening. For instance, changing to the wristwatch informs the user that a time-consuming action is taking place.

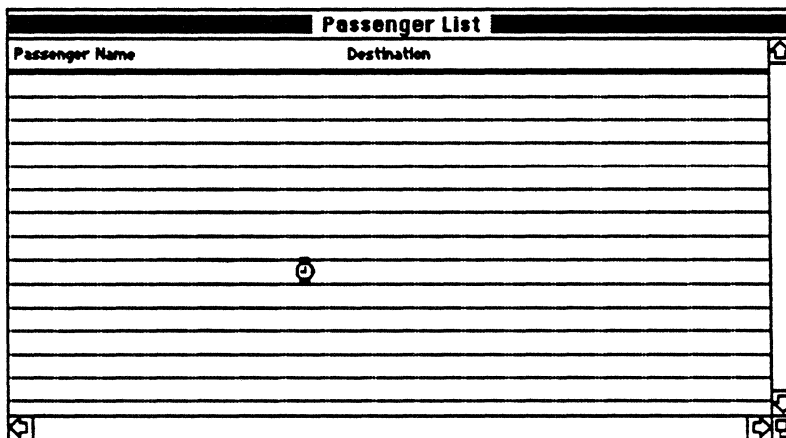
Several pointers are predefined by MWS. They are shown in Table 4-1 together with their resource IDs, which are used in the Set Cursor (C002) command. You can define your own pointers and save them as resources in the MWS document.

■ **Table 4-1** Cursor resource IDs

Image	Name	'CURS' resource ID
	Arrow	0
	I-beam	1
	Crosshair	2
	Crossbar	3
	Wristwatch	4

In the routine that sets up the Reservations dialog box, the Set Cursor command is used to change the pointer to a wristwatch while the dialog box is being created. Figure 4-2 shows the screen with the wristwatch pointer.

■ **Figure 4-2** Wristwatch pointer created with the Cursor Director



Let's examine the command in detail.

```
MWSPut ("C002", "4");
```

- C is the class for the Cursor Director.
- 002 is the identifier for the Set Cursor command.
- 4 is the alias for the cursor type, a wristwatch. The wristwatch shows that a lengthy procedure is taking place. The cursor image is set to the cursor specified by the *cursorID* parameter. The cursor will remain a wristwatch regardless of mouse movement.

This command sends the Auto-Track command (C001):

```
MWSPut ("C001", "");
```

The Auto-Track command returns control of the cursor to the MacWorkStation application.

---

## The Dialog Director

Dialog boxes are used to get information from the user. Using the mouse and keyboard, the user can adjust controls and enter data into editable text fields. Controls in dialog boxes appear as push buttons, check boxes, and radio buttons.

The Dialog Director lets the client application specify what the dialog box will look like, what items it should have, where the items should be placed, whether any items or the box itself should be displayed in color, and how items should behave. Any combination of text fonts, sizes, and styles is permissible within a dialog box. Other dialog box options include vertical and horizontal scroll bars, and size and zoom boxes. The Dialog Director also lets the client application save the dialog box as a resource in the MWS document. The client application can then open the dialog box by using its resource ID number, rather than reconstructing it.

Once the dialog box is constructed, the client application can have MWS do all the work of managing the dialog box while the user is making selections or entering text. When the user changes a control or enters text, MWS sends an event to the application with the user's selections and entries.

---

## Dialog item types

Your client application creates a dialog box by specifying a dialog box window, then adding dialog items to it. As each item is added, it is assigned an index number. The first item added is item 1, the second is item 2, and so on. Dialog items are referred to by this index number.

The client application adds a variety of dialog box item types. A dialog box can be as simple as a picture with an OK button, or very complex with a number of buttons, pictures, editable text fields, and scrolling-entries items. Table 4-2 lists the keywords for each of the dialog item types.

■ **Table 4-2** Keywords and dialog item types for the MacWorkStation Dialog Director

Keyword	Abbreviation	Dialog item type
BUTTON	B	Button
CHECK	C	Check box
EDIT	E	Editable text field
ICON	I	Icon
LINE	L	Line divider
MENU	M	Pop-up menu
PICTURE	P	Picture
RADIO	R	Radio button
SCROLL	S	Scrolling-entries item
TEXT	T	Static text field
USER	U	User item (custom control)

---

## Control items

Buttons, check boxes, radio buttons, and user items (custom controls) are control items. Control items are managed by the Macintosh User Interface Toolbox Control Manager. A control item can work like a switch (off and on), or a volume control (range). The control is set up with minimum, maximum, and current values. Users can adjust the control, changing the current value. A control has an indicator of some sort to show the current value. A standard control that works like a switch has only two values: 0 and 1. Check boxes and radio buttons are examples.

Another feature of controls is that they can be enabled or disabled. This capability is referred to as setting a control's highlighting, or its active state. An inactive control cannot be changed by the user—it appears to be dimmed and usually indicates no value.

## A dialog box routine

The messages below create the dialog box shown in Figure 4-3. The routine sets the cursor type, disables the Bear Cal menu, and sets up the title, fields, clusters, and buttons for the dialog box. It then returns control of the cursor to MWS.

```
MWSPut("C002","4");
MWSPut("D001","4;F;Bear Cal Reservation System;16;F;30,50,482,335");
MWSPut("D009","4; B; 380,10,440,30; OK; F; T; 1");
MWSPut("D009","4; B; 380,40,440,60; Cancel; T; F; 0");
MWSPut("D009","4; T; 10,10,110,26; Passenger:");
MWSPut("D009","4; E; 115,10,360,26; F; 1; T;");
MWSPut("D009","4; T; 10,35,110,51; Destination:");
MWSPut("D009","4; E; 115,35,360,51; F; 0; T; A; (");
MWSPut("D009","4; T; 10,60,110,76; Depart:");
MWSPut("D009","4; E; 115,60,185,76; F; 0; T;");
MWSPut("D009","4; T; 205,60,285,76; Return:");
MWSPut("D009","4; E; 290,60,360,76; F; 0; T;");
MWSPut("D009","4; L; 12,85;440,85;");
MWSPut("D009","4; T; 10,95,110,111; Airline:");
MWSPut("D009","4; E; 115,95,330,111; F; 0; T;");
MWSPut("D009","4; T; 10,120,110,136; Time:");
MWSPut("D009","4; T; 1345,1120,1400,1136; Hidden:");
MWSPut("D009","4; E; 115,120,330,136; F; 0; T;");
MWSPut("D009","4; E; 1405,1120,1440,1136; F; 0; T;");
MWSPut("D009","4; T; 10,145,110,161; Gate:");
MWSPut("D009","4; E; 115,145,330,161; F; 0; T;");
MWSPut("D009","4; T; 1235,1145,1295,1161; hidden:");
MWSPut("D009","4; E; 1300,1145,1440,1161; F; 0; T;");
MWSPut("D009","4; L; 12,170;440,170;");
MWSPut("D009","4; T; 370,180,440,196; Payment:");
MWSPut("D009","4; P; 375,240;101;F;F;1");
MWSPut("D009","4; P; 375,200;100;F;F;1");
MWSPut("D009","4; L; 360,175;360,275;");
MWSPut("D009","4; T; 10,180,110,196; Incidentals:");
MWSPut("D009","4; C; 15,200,110,216; Smoking; T; T; 2; 0; 0;");
MWSPut("D009","4; C; 15,220,110,236; Dinner; T; T; 2; 0; 0;");
MWSPut("D009","4; C; 15,240,110,256; Movie; T; F; 2; 0; 0;");
MWSPut("D009","4; C; 15,260,110,276; Rental Car; T; T; 2; 0; 0;");
MWSPut("D009","4; T; 130,180,230,196; Class:");
MWSPut("D009","4; R; 135,200,230,216; First; T; F; 3; 0; 0;");
MWSPut("D009","4; R; 135,220,230,236; Business; T; F; 3; 0; 0;");
MWSPut("D009","4; R; 135,240,230,256; Coach; T; F; 3; 0; 1;");
MWSPut("D009","4; T; 250,180,350,196; Seating:");
MWSPut("D009","4; R; 255,200,350,216; Window; T; F; 4; 0; 0;");
MWSPut("D009","4; R; 255,220,350,236; Center; T; F; 4; 0; 1;");
MWSPut("D009","4; R; 255,240,350,256; Aisle; T; F; 4; 0; 0;");
MWSPut("D007","4");
MWSPut("C001","");
```

■ **Figure 4-3** Reservations dialog display with clusters

**Bear Cal Reservation System**

Passenger:

Destination:

Depart:  Return:

---

Airline:

Time:

Gate:

---

<b>Incidentals:</b>	<b>Class:</b>	<b>Seating:</b>	<b>Payment:</b>
<input type="checkbox"/> Smoking	<input type="radio"/> First	<input type="radio"/> Window	
<input type="checkbox"/> Dinner	<input checked="" type="radio"/> Coach	<input checked="" type="radio"/> Center	
<input type="checkbox"/> Movie	<input type="radio"/> Business	<input type="radio"/> Aisle	
<input type="checkbox"/> Rental Car			

The following line of code is the first command using the Dialog Director:

```
MWSPut("D001", "4;F;Bear Cal Reservation System;16;F;30,50,482,335");
```

- D is the class for the Dialog Director.
- 001 is the identifier for the command New Dialog.
- 4 is the alias for this particular dialog box.
- F is a flag indicating that the dialog box is not modal.
- Bear Cal Reservation System is the title of the dialog window.
- 16 signifies the shape of the window, as explained in "The Window Director," later in this chapter.
- F is a flag that denotes that the dialog box does not appear when it is created. The command Show Dialog (D007) actually draws the dialog box.
- 30, 50, 482, 335 specifies in global screen coordinates the dimensions of the dialog box.

## Clustering items

**Clustering** allows dialog box items to be grouped together. For example, if you cluster radio buttons, then only one button at a time can be selected. When the user clicks a button that is not selected, it becomes selected and the other one is deselected. You specify an item's cluster in the Add Item (D009) command. Giving an item a cluster identifier of zero (0) indicates that it does not belong to a cluster.

Here is an Add Item message that assigns the item to a cluster:

```
MWSPut("D009", "4; C; 15,220,110,236; Dinner; T; T; 2; 0; 0;");
```

- D is the class for Dialog Director.
- 009 is the identifier for the command Add Item.
- 4 is the alias for this dialog box.
- C is an abbreviation for the item type, CHECK, which adds a check box to the list.
- 15, 220, 110, 236 specifies in local window coordinates the rectangle in which the item will be drawn.
- Dinner is the title that will appear to the right of the control item.
- The active flag is true (T), enabling the check box for the user to check.
- The report flag is true (T), so an event will be generated when the user clicks in the control.
- 2 is the cluster identifier. Since the identifier is not 0, the item is part of the cluster with other items that have a 2 identifier in this position.
- The oversee parameter is 0. This value indicates that the item does not oversee a cluster. Overseeing is explained fully in the *MacWorkStation Programmer's Reference*.
- Valid values for the last item are 0 and 1. The 0 means that the check box is not checked.

The following lines illustrate the use of clusters in dialog boxes.

```
MWSPut("D009", "4; T; 130,180,230,196; Class:");
MWSPut("D009", "4; R; 135,200,230,216; First; T; F; 3; 0; 0;");
MWSPut("D009", "4; R; 135,220,230,236; Business; T; F; 3; 0; 0;");
MWSPut("D009", "4; R; 135,240,230,256; Coach; T; F; 3; 0; 1;");
```

This series of messages sets up a text label, `Class:`, for three radio buttons. The parameter immediately after each of the class titles—`First`, `Business`, and `Coach`—of the three radio buttons is the **active flag**. This flag is true for each button, indicating that the button is enabled when the dialog box is displayed. Notice that all of the buttons have the same cluster identifier (3). When the user clicks one of the radio buttons, it will be selected and given a value of 1. Because the radio buttons are a cluster, the other items will be deselected and their values set to zero (0). `Coach` is the default for this cluster, which is set by the last parameter.

## The File Director

The File Director controls the exchange of information between the client application and files on the Macintosh computer. The File Director allows the client application to create and open files and access the information they contain. Your application can also delete and rename files, open and close either fork of a Macintosh file, read and write data, and get or set information about the files. Your application may lock specific byte ranges within a file to prevent changes, and can specify the permission level for files on AppleShare® volumes.

The client application can get information about volumes, including the volume name, free blocks available, and the number of files on the volume. It can also get or set access privileges for directories of AppleShare volumes.

A Macintosh file consists of two file forks, called the data fork and the resource fork. When a MacWorkStation file is created, both forks are created, each with a logical size of zero (0). When a file is deleted, both forks are deleted. However, the forks behave as individual files in all other respects.

A fork is a finite sequence of numbered bytes. The first byte is byte zero (0), the second byte is one (1), and so on, up to the logical end-of-file. The current position, or **mark**, is the number of the next byte that will be written or read. The mark automatically moves forward one position for every byte read or written. If the mark reaches the logical end-of-file while writing, both the mark and the logical end-of-file are moved forward one position for every byte written to the file. The value of the mark can never exceed the value of the logical end-of-file.

Normally, client applications will operate only on the **data fork**. The data fork is often used for storing ASCII text, such as that generated by MWS in text and list windows. The **resource fork** is used to store resources. Ordinarily, you would not change individual bytes of the resource fork. You would use the File Director to copy an entire resource fork.

This routine first creates an Excel text file. It then generates data that is in turn written to the file's data fork. The data is then read back in and written to the Flight Schedule window.

```
MWSPut ("F001", "Schedule;XCEL;TEXT");      /* Create new file */
MWSPut ("F004", "1;Schedule");              /* Open file data fork */
WriteFlight ();
MWSPut ("F006", "1");                      /* Close file data fork */
```

Here is an explanation of the first message:

- **F** is the class, indicating the File Director.
- **001** is the identifier for the command Create File/Folder.
- **Schedule** is the filename.
- **XCEL** is the four-character creator.
- **TEXT** is the four-character file type.



---

## The Graphics Director

The Graphics Director is used to create images in graphics windows. With the Graphics Director, the client application can draw lines, geometric shapes, text, icons, and complicated pictures.

A MacWorkStation graphics window "remembers" what has been drawn, so the client application does not have to redraw the image should its window be hidden temporarily.

---

### Coordinates in graphics windows

Many of the graphics commands require values to specify where to draw. These values represent points or rectangles in the coordinates of the window's drawing plane. The left-top corner of the window's drawing plane is the point (0,0). The drawing plane is limited to the size of one U.S. letter-sized printed page. This page translates to a drawing rectangle 576 pixels wide by 720 pixels tall.

The window's content area is the visible area of the window where drawing occurs. When a graphics window is created, the left-top corner of the content area is set to the left-top corner of the drawing plane (0,0). When a window is scrolled, the left-top corner of the content area changes, while the coordinate system of the drawing plane remains the same. Because the graphics commands are specified in the coordinate system of the drawing plane, the client application doesn't have to deal with scrolling; MWS handles it.

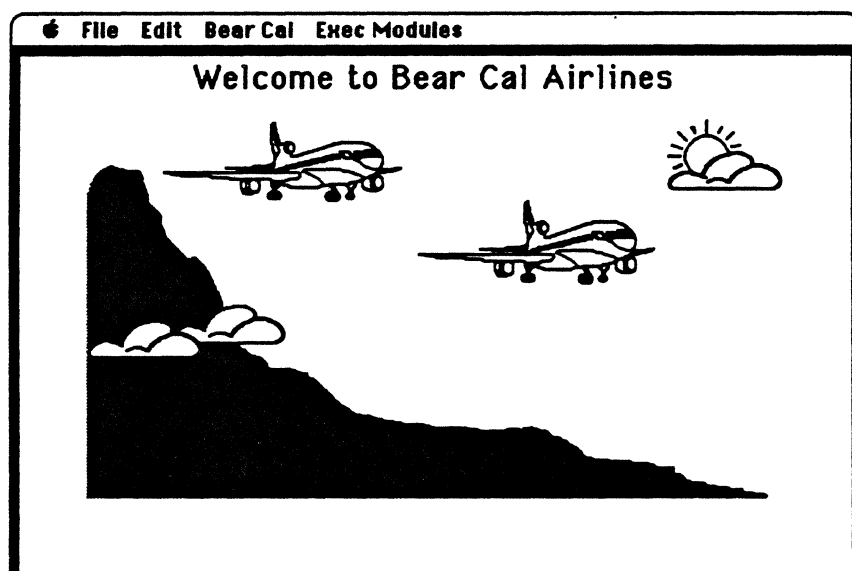
On windows without scroll bars, the coordinates of the window's content area always coincide with the window's drawing plane. Because the user will not be able to scroll the window, you should be careful not to draw outside the window's content area.

## Graphics commands

This listing illustrates the use of several graphics commands. They create the "Welcome to Bear Cal Airlines" display shown in Figure 4-4.

```
MWSPut ("W001", "8;G;;2;F;F;5,25,507,340;0;0");  
MWSPut ("G024", "8;40,40;1500");  
MWSPut ("G024", "8;380,40;1000");  
MWSPut ("G014", "8;Geneva");  
MWSPut ("G015", "8;18");  
MWSPut ("G016", "8;B");  
MWSPut ("G008", "8;105,20");  
MWSPut ("G018", "8;Welcome to Bear Cal Airlines");  
MWSPut ("W003", "8");
```

■ **Figure 4-4** Screen display created with the Graphics Director



Here is a detailed description of the first two Graphics Director messages.

```
MWSPut ("G024", "8;40,40;1500");
```

- **G** is the class of the Graphics Director.
- **024** is the identifier for the Draw Picture command.
- **8** is the window alias.
- **40, 40** are the coordinates for the left-top corner of the picture.
- **1500** is the resource ID of the 'PICT' resource used for the graphics. This 'PICT' resource contains the airplanes and some of the clouds.

## APPLE CONFIDENTIAL

```
MWSPut ("G024", "8;380,40;1000");
```

- 380, 40 are the coordinates for the left-top corner of the picture.
- 1000 is the resource ID for this 'PICT'. This 'PICT' contains the graphic of the clouds with the sun.

The last four lines in the example create the banner "Welcome to Bear Cal Airlines." Let's look at each line in detail.

```
MWSPut ("G015", "8;18");
```

- 015 is the command to Set Text Size.
- 8 is the alias for the window.
- 18 is the point size of the text.

```
MWSPut ("G016", "8;B");
```

- 016 is the Set Text Style command.
- 8 is the alias for the window.
- B is the string name for the style, in this case boldface.

```
MWSPut ("G008", "8;105,20");
```

- 008 is the Set Pen Location command.
- 105, 20 are the coordinates for the left-top corner of the banner.

```
MWSPut ("G018", "8;Welcome to Bear Cal Airlines");
```

- 018 is the Draw Text command. It draws text in the window with the alias of 8. The text is Welcome to Bear Cal Airlines.

## The List Director

The List Director allows the client application to display textual data in a variety of list formats, and provides editing facilities for the user. The client application can use list windows to present information in tables or spreadsheet format. The list might be a collection of items the user can choose from, or information the user is looking for. Figure 4-5 shows an example of a list window.

■ **Figure 4-5** Flight schedule created with the List Director

Airline	Time	Gate
American Airlines	1:50 am	37
Bear Cal Airlines	10:20 am	1
Bear Cal Airlines	5:35 am	4
Continental Airlines	11:50 am	12
Continental Airlines	2:50 pm	13
Eastern Airlines	4:43 pm	19
Republic Airlines	7:50 am	22
Republic Airlines	1:30 pm	23
United Airlines	10:50 am	14

Both the client application and the user can add, delete, and change the contents of individual records and fields in the list. The user can select, cut, copy, and paste records and fields.

The client application can set titles over list columns, set tab stops for the columns, and choose display options such as fonts, font sizes, and grid lines. The user can choose the font and size of text in the list window, if the application provides the standard Font and Size menus.

The list window can be printed or saved to a document. The document is created as a Microsoft Excel-text-formatted file. The contents of the file are ASCII text with fields separated by horizontal tab characters (ASCII \$09), and with records separated by carriage returns (ASCII \$0D). This format is compatible with most spreadsheet and database programs.

Here is a routine that creates a new flight schedule window, like the one in Figure 4-5, and uses List Director commands to set appropriate tabs and titles:

```
MWSPut ("W001", "9;L;FLIGHT SCHEDULE;0;F;F;20,75,492,315;0;0");
MWSPut ("L019", "9;1,200;2,350");          /*      Tab breaks      */
MWSPut ("L016", "9;Airline\tTime\tGate");
```

The first message is a Window Director command that creates a new list window called `FLIGHT SCHEDULE` with the alias of `9`.

- `L` indicates that the kind of window is List.
- `0` specifies the shape of the window.
- The first `F` means that the close box flag is false, so there is no close box in the leftmost corner of the title bar.
- The second `F` means that the visible flag is false, so the window is initially invisible.
- The coordinates for the location of the window are `20, 75, 492, 315`.
- `0` indicates that the window uses the default menu bar.
- The cursor parameter is `0`, indicating that a crossbar cursor will appear, which is the default for List windows.

The next two messages are List Director commands.

- The command `L019` is Set Tab Stops. In this case, the tab stop coordinates are `200` for tab stop `1` and `350` for tab stop `2`.
- Command `L016` is Set Headings. It allows the client application to place titles above the columns in the window. In this case, the titles are `Airline`, `Time`, and `Gate`.

## The Menu Director

The Menu Director lets you create menus. In Macintosh applications, menu titles appear across the top of the screen in the menu bar. The Macintosh user simply positions the cursor over a menu title in the menu bar and presses the mouse button. The menu then appears, displaying the list of menu items. As long as the mouse button is held down, the menu is displayed. Dragging the mouse over the menu items causes each item to be highlighted in turn. If the mouse button is released over an item, that item is "chosen."

The client application can instruct MWS to display any of the standard Macintosh menus: the File, Edit, Font, Size, Search, and Display menus. The Apple menu is always displayed. The client application can also display menus it defines either through commands or as resources, including hierarchical and color menus. To add menus containing commands unique to your application, you can send strings describing the titles of the menus and their items. Figure 4-6 shows a sample custom menu. You can also use menus that are stored as resources in the MWS document.

MWS manages all the menu items (commands) that the user chooses from the standard menus. However, when the user chooses an item from a client application menu, the selection is sent to the client application as an event by MWS.

### ■ Figure 4-6 Pull-down menu example

Bear Cal	
Reservations	⌘R
Show Flight Schedule	⌘F
Hide Log	⌘L
About Bear Cal	⌘A

The following messages create the menu in Figure 4-6.

```
MWSPut("M004","0;Bear Cal,1,Reservations/R,Show Flight Schedule/F,(-,");
MWSPut("M006","0;1;Hide Log/L,(-,About Bear Cal/A");
```

- **M** is the class for the Menu Director.
- **004** is the identifier for the Add Menus to Menu Bar command.
- **Zero (0)** is the alias for the default menu bar.
- **FILE** and **EDIT** are keywords for standard menus.
- **Bear Cal**, in the second command, is a custom menu name.
- The alias for this particular menu is **1**.
- **Reservations** and **Show Flight Schedule** are the first menu items listed in the **Bear Cal** menu. The parenthesis and hyphen in the list will display a disabled, dashed line after the menu items. The Command-key equivalents for the menu items are **/R**, **/F**, **/L**, and **/A**.

- M006 is the Append Items to Menu command. This command adds the Hide Log and About Bear Cal items to the menu. The Command-key equivalents for these menu items are /L, and /A. Another disabled, dashed line is displayed in the menu.

Consider this message:

```
MWSPut ("M008", "0;1;1");
```

This Menu Director command enables a menu item so that it can be selected by the user.

- 008 is the Enable Item command.
- Zero (0) refers to the default menu bar, the first 1 refers to the Bear Cal menu, and second 1 is the item to enable.

If the item was already enabled, Enable Item does nothing.

The Delete Menu From Menu Bar (M005) command allows your client application to delete from the menu bar all menus except the Apple menu. You can append items to a menu by using Append Items to Menu (M006). You might also want to add a check mark to the left of a menu item's text, to denote the status of an item or of the mode it controls. You can add a check mark with the Check Item and Exclusively Check Item commands. See the *MacWorkStation Programmer's Reference* for a complete discussion of each command.

---

## The Process Director

The Process Director handles program control and administration. Such tasks include protocol **handshaking**, getting machine and software versions, setting user wait states, and quitting the Macintosh application.

The following message sends the Host Online (P001) command:

```
MWSPut ("P001", "");
```

The client application must send this command at the beginning of each session in response to the MacWorkStation Online (P256) event. The MacWorkStation Online event is sent after the MWS document is opened and its CCL script logs on to the client application.

---

## The Text Director

The Text Director handles the contents of text windows. It allows the user to view, scroll, edit, save, and print text windows. The client application may call on the Text Director to transfer text to and from a text window, or to alter the window display. It can select a specific range of text; locate text; specify mixed fonts, font sizes, and styles; and find out text characteristics. Text windows can hold up to 32K of text.

## The Window Director

The Window Director lets the client application create and manipulate windows on the Macintosh computer screen. Through other MacWorkStation directors, text and graphic information can be exchanged between the client application and the user. A window's contents can be retrieved from or saved to Macintosh documents in formats accepted by other Macintosh applications.

When a window is created, it is assigned a window kind that determines which MWS director will control it. MWS supports graphics, text, and list windows. Also, a client application can create windows by using parameters stored in the 'WIND' resource.

### Window kinds

The window kind specifies which MacWorkStation director is responsible for the contents of the window. When the client application creates a window, the window kind is specified as a keyword parameter. The client application then manipulates the contents of the window by making calls to the corresponding director. A window's kind cannot be changed.

For example, the client application creates a graphics window by specifying the window kind as **GRAPHICS**. Then the client application makes calls to the Graphics Director to draw graphics into the window.

The window kind also specifies the type of Macintosh document used when saving the contents of the window. MWS supports file formats for three widely used applications: MacDraw, MacWrite, and Microsoft Excel.

Table 4-3 lists the keywords for the window kinds, their corresponding document kinds, and the directors that manage the windows. The creator and type are Macintosh file components, described in "File Director," in Chapter 2 of the *MacWorkStation Programmer's Reference*.

■ Table 4-3 Window kinds

Keyword	Abbreviation	Document kind	Creator	Type	Director
GRAPHICS	G	MacDraw	MDRW	PICT	Graphics Director
LIST	L	Excel Text	XCEL	TEXT	List Director
TEXT	T	MacWrite Text	MACA	TEXT	Text Director
RESGRAPHICS	RG	MacDraw	MDRW	PICT	Graphics Director
RESLIST	RL	Excel Text	XCEL	TEXT	List Director
RETEXT	RT	MacWrite Text	MACA	TEXT	Text Director

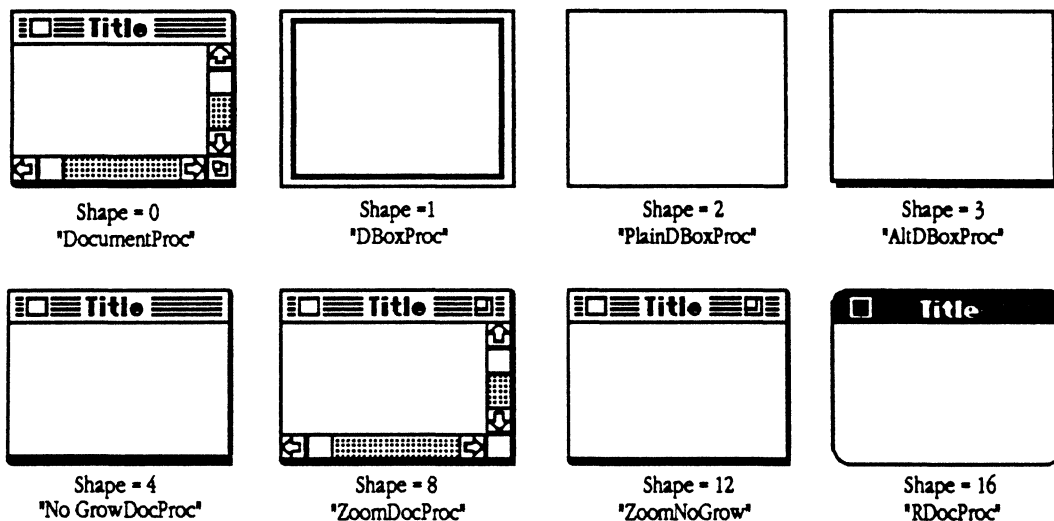
If the window kind is **RESLIST**, **RESPICT**, or **RETEXT**, then MWS looks for a 'WIND' resource in the document. You can use this 'WIND' resource to specify the window's title, shape, size, whether it has a closebox, and whether it is visible.



## Window shapes

Windows can have different shapes, sizes, controls, and characteristics. A window may or may not have a title bar. It may have a close box where the user clicks to close the window. It may also have a zoom box, where the user clicks to zoom the window. You specify the window shape, size, title, and other characteristics when you create it. Figure 4-7 shows the various shapes of windows.

■ Figure 4-7 Window shapes



The default diameter of the curvature of a rounded-corner window is 16 pixels. Smaller and greater curvatures can be specified, ranging from 17 to 23 pixels.

---

## Positioning windows

When you create a window, you give the rectangular coordinates where the window will be drawn on the Macintosh screen. The left-top corner of the Macintosh screen is the point (0,0). As with other applications, MWS allows several windows to be open at the same time. The frontmost window is the only active window at any given time. The active window is the one in which the user is currently working. Windows can be on top of other windows, so that any window may partially or completely obscure the others. Windows with title bars can be moved by the user in order to see hidden windows. If the user clicks in a window, it becomes the frontmost, active window.

---

## Window options

Window management is normally handled by MWS. By using the standard menus, the client application lets the user create, open, close, save, and print windows, edit text, and change fonts. However, your application can control this behavior by setting the window options. You also use window options to have MWS inform the client application when a user has activated, deactivated, moved, or resized a window.

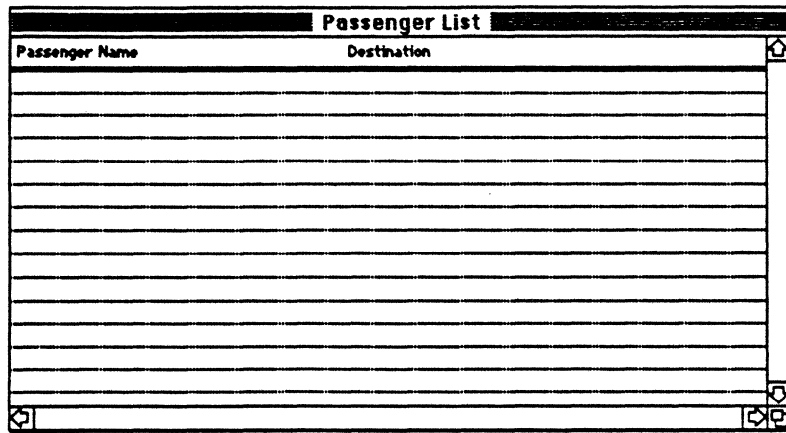
MacWorkStation window options can be set or cleared for an existing window or as defaults for new windows as they are created. The window options are listed in Table 4-5.

■ Table 4-4 Window options

Option	Code	Default	Description
All	0	T	TRUE: All window options are TRUE. FALSE: All window options are FALSE.
Saveable	1	T	TRUE: A user can save the contents of the window to disk. FALSE: A user cannot save the contents of the window.
Printable	2	T	TRUE: A user can print the contents of the window. FALSE: A user cannot print the contents of the window.
Fonts	3	T	TRUE: A user can change the font and point size of text. FALSE: A user cannot change the font and point size.
Editable	4	T	TRUE: A user can edit the contents of the window, if the Edit menu is installed. FALSE: A user cannot edit the contents of the window.
Copyable	5	T	TRUE: A user can copy the contents of the window if the Edit menu is installed. FALSE: A user cannot copy the contents of the window.
Selectable	6	T	TRUE: A user can select part of the window's contents. FALSE: A user cannot select part of the window's contents, only the window's entire contents.
Closeable	7	T	TRUE: A user can close the window by choosing Close from the File menu, or clicking in the window's close box. FALSE: A user cannot close the window.
Dismiss	8	T	TRUE: MWS closes windows without sending an event. FALSE: MWS sends an event when the user chooses Close from the File menu, or clicks in the window's close box.
Activate	9	F	TRUE: MWS sends an event when a window is activated. FALSE: MWS does not send the event.
Deactivate	10	T	TRUE: MWS sends an event when a window is deactivated. FALSE: MWS does not send the event.
Move	11	F	TRUE: MWS sends an event when a window is moved. FALSE: MWS does not send the event.
Resize	12	F	TRUE: MWS sends an event when a window is resized. FALSE: MWS does not send the event.
Mixed Styles	13	F	TRUE: A user can have mixed text characteristics in the window. FALSE: A user cannot have mixed text characteristics. All text must be the same font, point size, text style, and color.

Figure 4-8 shows a list window created with the Window Director.

■ **Figure 4-8** List window created with the Window Director



The following message creates the window shown in Figure 4-8:

```
MWSPut ("W001", "7;L;Passenger List;0;F;F;20,75,492,315;0;0");
```

- `w` is the class for the Window Director.
- `001` is the identifier for the command New Window.
- `7` is the alias of this window.
- `L` means that this is a list window.
- `Passenger List` is the title of the window.
- `0` indicates the shape of the window, as shown in Figure 4-7.
- The first `F` is the close box flag. In this case it is false, so there is no close box in the window.
- The next `F` is the visible flag, which in this case is false, so the window is invisible until the Show Window command makes it visible.
- `20, 75, 492, 315` are the coordinates for the location of the window.
- `0` specifies which menu bar should be shown when the window is the frontmost window.
- The next parameter, `0`, specifies which cursor is displayed when the mouse pointer is over the content region of the active window. In this case, the cursor is the crossbar, which is the default.

This message calls the Toss Window (`w002`) command to dispose of the window created in the previous example:

```
MWSPut ("W002", "7");
```

---

## The Exec Director

The Exec Director allows you to use executable code modules (exec modules). You can create an exec module to perform operations unique to your client application. An exec module is stored in the MWS document as a 'CODE' resource. (See *MacWorkStation Programmer's Reference* for more information about exec modules.) These messages load and invoke an exec module:

```
MWSPut ("X001", "1;");  
MWSPut ("X003", "1;");
```

The first command, Load Exec Module (X001), places the exec module's code in the Macintosh computer's memory. The Invoke Exec Module (X003) command executes the module's code. An exec module does not start running until it is invoked. An exec module remains in memory until it is unloaded. This message unloads the exec module:

```
MWSPut ("X002", "1;");
```



## Appendix **"Bear Cal" Program Source Code**

THIS APPENDIX CONTAINS the source code for "Bear Cal," a demonstration client application written in MPW C 3.0. Bear Cal is a fictional airline. This client application handles the airline's reservations. ■

## APPLE CONFIDENTIAL

```
/*
 *
 *      Bear Cal Demo
 *
 *      "Bear Cal" is a simple client application that uses many of the
 *      features of MWS. A version of this program is supplied on the
 *      MacWorkStation Samples Disk, and may have more recent changes.
 *
 */
```

```
#include <stdio.h>
#define false 0
#define true 1
```

```
static int AllDone = false;
static int ExecOpen = false;
static int DlgRes = false;
static int OnPayment = false;
static int SecondPayment = false;
static int OnShow = true;
static int OnFlight = false;
```

```
static char *FlightStr[] = {
    "American Airlines\t1:50 am\t37\r",
    "Bear Cal Airlines\t10:20 am\t1\r",
    "Bear Cal Airlines\t5:35 am\t4\r",
    "Continental Airlines\t11:50 am\t12\r",
    "Continental Airlines\t2:50 pm\t13\r",
    "Eastern Airlines\t4:43 pm\t19\r",
    "Republic Airlines\t7:50 am\t22\r",
    "Republic Airlines\t1:30 pm\t23\r",
    "United Airlines\t10:50 am\t14\r"
};
```



APPLE CONFIDENTIAL

```
static char *Flight1[] = {  
    "American Airlines",  
    "Bear Cal Airlines",  
    "Bear Cal Airlines",  
    "Continental Airlines",  
    "Continental Airlines",  
    "Eastern Airlines",  
    "Republic Airlines",  
    "Republic Airlines",  
    "United Airlines"  
};
```

```
static char *Flight2[] = {  
    "1:50 am",  
    "10:20 am",  
    "5:35 am",  
    "11:50 am",  
    "2:50 pm",  
    "4:43 pm",  
    "7:50 am",  
    "1:30 pm",  
    "10:50 am"  
};
```

```
static char *Flight3[] = {  
    "37",  
    "1",  
    "4",  
    "12",  
    "13",  
    "19",  
    "22",  
    "23",  
    "14"  
};
```

## APPLE CONFIDENTIAL

```
/*
 * The Main event loop of the program.
 */
*****/
```

```
main()

{
    char    EvtClass;
    int     EvtID;
    char    EvtParms[256];

    MWSInit();

    do {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        switch(EvtClass) {
            case 'A': DoAEvent(EvtID, EvtParms);
                      break;
            case 'D': DoDEvent(EvtID, EvtParms);
                      break;
            case 'F': DoFEvent(EvtID, EvtParms);
                      break;
            case 'G': DoGEvent(EvtID, EvtParms);
                      break;
            case 'L': DoLEvent(EvtID, EvtParms);
                      break;
            case 'M': DoMEvent(EvtID, EvtParms);
                      break;
            case 'P': DoPEvent(EvtID, EvtParms);
                      break;
            case 'T': DoTEvent(EvtID, EvtParms);
                      break;
            case 'W': DoWEvent(EvtID, EvtParms);
                      break;
            case 'X': DoXEvent(EvtID, EvtParms);
                      break;
        }
    } while (!AllDone);

    MWSQuit();
}
```

## APPLE CONFIDENTIAL

MWSInit()

```
    /*****
    *   Do anything special to start the program.   *
    *****/
{
    printf("%s", "GO BEARCAL\n");    /* Signal to MWS CCL script */
}
```

MWSQuit()

```
    /*****
    *   Do anything special to stop the program.   *
    *****/
{
}
```

DoAEvent (EvtID, EvtParms)

```
    int     EvtID;
    char    *EvtParms;

    /*****
    *   Process Alert Director events.   *
    *****/
{
}
```

# APPLE CONFIDENTIAL

DoDEvent (EvtID, EvtParms)

```

int      EvtID;
char     *EvtParms;

/*****
 *   Process Dialog Director events.
 *****/

(
char  Alias[10];
char  ItemNum[10];
char  OnStatus;
int   Counter = 0, Count;
int   StartItem, StartOn;
int   i= 0, j= 0;

while (EvtParms[Counter] != ';')      /* Put Alias into array */
    Alias[i++] = EvtParms[Counter++];
i--;                                  /* Put Alias length into i */

StartItem = ++Counter;

while (EvtParms[Counter] != ';')      /* Put ItemNum into array */
    ItemNum[j++] = EvtParms[Counter++];
j--;                                  /* Put ItemNum length into j */

Counter += 3;
StartOn = Counter;
OnStatus = EvtParms[Counter];

if (EvtID == 257) {                  /* Control item pressed */
    if ((Alias[0] == '4') || (Alias[0] == '5') ||
        (Alias[0] == '6') || (Alias[0] == '8')) {
        switch (ItemNum[0]) {
            case '1':                /* OK button */
                if (Alias[0] == '4')
                    ProcOK();
                else if (Alias[0] == '5')
                    AnswerSC();
                else if (Alias[0] == '6')
                    AnswerGCC();
                break;

```

# APPLE CONFIDENTIAL

```
case '2':
    if (j > 0) {
        switch (ItemNum[1]) {
            case '4': ProcGCC();
                        break;
            case '5': ProcSC();
                        break;
            case '8':
                if (OnStatus == '1')
                    ProcSmoke();
                else
                    ProcNotSmoke();
                break;
            case '9':
                if (OnStatus == '1')
                    ProcDinner();
                else
                    ProcNoDinner();
                break;
        }
    } else {
        if (Alias[0] == '4')
            ProcCancel();
        if (Alias[0] == '8')
            ProcRentOK();
    }
    break;

case '3':
    if (j > 0) {
        switch (ItemNum[1])
            case '1':
                if (OnStatus == '1')
                    ProcRent();
                else
                    ProcNoRent();
                break;
    } else {
        if (Alias[0] == '8')
            ProcRentCancel();
    }
    break;
}
}
}
```

## APPLE CONFIDENTIAL

DoFEvent(EvtID, EvtParms)

```
int    EvtID;
char   *EvtParms;
```

```
/******
 *   Process File Director events.
 *****/
```

```
{
}
```

DoGEvent(EvtID, EvtParms)

```
int    EvtID;
char   *EvtParms;
```

```
/******
 *   Process Graphic Director events.
 *****/
```

```
{
}
```

# APPLE CONFIDENTIAL

DoLEvent (EvtID, EvtParms)

```

int      EvtID;
char     *EvtParms;

/*****
*   Process List Director events.
*****/

{
char     RecNum[10];
int      recno;
char     str[255];

if (EvtID == 256) {          /* User double-clicked a record */
    if (EvtParms[0] == '7') {
        strncpy(&RecNum,&EvtParms[2],(strlen(&EvtParms[2]) - 1));
        RemoveFromLog(RecNum);          /* Removes record from Log */
    }
    if (EvtParms[0] == '9') {          /* Put flight information into
                                        dialog box */
        recno = EvtParms[2] - '0' - 1;
        MWSPut("D011",strcat(strcpy(str,"4;13;"),Flight1[recno]));
        MWSPut("D011",strcat(strcpy(str,"4;16;"),Flight2[recno]));
        MWSPut("D011",strcat(strcpy(str,"4;19;"),Flight3[recno]));
        OnFlight = !OnFlight;
        DisposeFlight();
        MWSPut("M007","0;1;2;Show Flight Schedule");
    }
}
}

```

# APPLE CONFIDENTIAL

DoMEvent (EvtID, EvtParms)

```

int      EvtID;
char     *EvtParms;

/*****
 *   Process Menu Director events.
 *****/

{
    char    MenuBar;
    char    MenuID;
    char    MenuItem;
    int     error = false;

    MenuBar = EvtParms[0];
    MenuID  = EvtParms[2];
    MenuItem = EvtParms[4];

    if ((MenuBar == '0') && (MenuID == '1')) {
        switch (MenuItem) {
            case '1':
                DoReservations();
                break;
            case '2':
                OnFlight = !OnFlight;
                if (OnFlight) {
                    DoFlightPrep();           /* Get new window */
                    error=DoFlight();         /* Write to window */
                    if (!error) {
                        ShowFlight();         /* Show flight window */
                        MWSPut("M007","0;1;2;Hide Flight Schedule");
                    }
                } else {
                    DisposeFlight();
                    MWSPut("M007","0;1;2;Show Flight Schedule");
                }
                break;
        }
    }
}

```



## APPLE CONFIDENTIAL

```
case '4':                                /* Show and Hide Log */
    OnShow = !OnShow;
    if (OnShow) {
        ShowLog();
        MWSPut("M007","0;1;4;Hide Log");
    } else {
        HideLog();
        MWSPut("M007","0;1;4;Show Log");
    }
    break;
case '6':
    AboutBearCal();
    break;
}

}

if ((MenuBar == '0') && (MenuID == '2')) {    /* Load exec module */
    if (ExecOpen) {
        MWSPut("X002","1;");
        MWSPut("M007","0;2;1;Start Example");
    } else {
        MWSPut("X001","1;");
        MWSPut("X003","1;");
        MWSPut("M007","0;2;1;Stop Example");
    }
    ExecOpen = !ExecOpen;
}

}
```

## APPLE CONFIDENTIAL

DoPEvent (EvtID, EvtParms)

```
int      EvtID;
char     *EvtParms;

/*****
 *   Process Process Director events.
 *****/
{
    switch (EvtID) {
        case 256 :    Startup();
                      break;
        case 257 :    AllDone = true;
                      break;
        case 258 :    Quit();
                      break;
    }
}
```

DoTEvent (EvtID, EvtParms)

```
int      EvtID;
char     *EvtParms;

/*****
 *   Process Text Director events.
 *****/
{
}
```

## APPLE CONFIDENTIAL

DoWEvent (EvtID, EvtParms)

```
int    EvtID;
char   *EvtParms;
```

```
/*
 * Process Window Director events.
 */
*****/
```

```
{
}
```

DoSEvent (EvtID, EvtParms)

```
int    EvtID;
char   *EvtParms;
```

```
/*
 * Process Exec Director events.
 */
*****/
```

```
{
}
```

# APPLE CONFIDENTIAL

Startup()

```

/*****
*   This routine is called once at the start of the program.
*   It sets up the menu bar, splash screen graphics,
*   Passenger Log window, the Meal Selection Dialog,
*   and the Special Meal window.
*****/

{
    MWSPut("P001","");
    MWSPut("P013","F;1");
    MWSPut("M001","0");
    MWSPut("M004","0;FILE;EDIT");
    MWSPut("M004","0;Bear Cal,1,Reservations/R,Show Flight Schedule/F,(-,");
    MWSPut("M006","0;1;Hide Log/L,(-,About Bear Cal/A");
    MWSPut("M004","0;Exec Modules,2,Start Example");
    MWSPut("M003","0");
    MWSPut("W001","7;L;Passenger List;0;F;F;20,75,492,315;0;0");
    MWSPut("L019","7;1,200;2,350");          /* Tab breaks */
    MWSPut("L015","7;T;4");                  /* Draws Horizontal grid
                                           lines in Log */
    MWSPut("L016","7;Passenger Name\tDestination\tDeparture Date");

    AboutBearCal();

    MWSPut("D001","3;T;Dinner;3;F;106,82,406,282");
    MWSPut("D009","3;T;20,20,280,45;Please Choose A Meal:");
    MWSPut("D009","3;R;30,60,150,80;Chicken Dinner;T;F;1;0;1");
    MWSPut("D009","3;R;30,90,150,110;Fish Dinner;T;F;1;0;0");
    MWSPut("D009","3;R;30,120,150,140;Special Meal;T;T;1;0;0");
    MWSPut("D009","3;B;220,60,270,85;OK;T;F;2");
    MWSPut("D009","3;B;220,120,270,145;Cancel;T;F;2");

    MWSPut("M001","2");
    MWSPut("M004","2;EDIT;FONT;SIZE");
    MWSPut("W001","3;T;Special Meal;0;T;F;50,50,462,314;2;1");
    MWSPut("T004","3;12");
    MWSPut("T003","3;Chicago");
    MWSPut("T005","3;Bold");
    MWSPut("W010","3;F;8");

    ShowLog();
}

```

# APPLE CONFIDENTIAL

AboutBearCal()

```

/*****
*   This routine supplies the calls to set-up the graphics used in   *
*   the AboutBearCal graphics window.                               *
*****/

{
    MWSPut("C004","",);
    MWSPut("W001","8;G;;2;F;F;5,25,507,340;0;0");
    MWSPut("G024","8;40,40;1500");
    MWSPut("G024","8;380,40;1000");

    MWSPut("G014","8;Geneva");
    MWSPut("G015","8;18");
    MWSPut("G016","8;B");
    MWSPut("G008","8;105,20");
    MWSPut("G018","8;Welcome to Bear Cal Airlines");

    MWSPut("W003","8");

    sleep(10);

    MWSPut("W002","8");
    MWSPut("C003","",);
}

```

DoFlightPrep()

```

/*****
*   This routine gets a new flight schedule window and sets       *
*   appropriate tabs and titles.                                   *
*****/

{
    MWSPut("W001","9;L;FLIGHT SCHEDULE;0;F;F;20,75,492,315;0;0");
    MWSPut("L019","9;1,200;2,350");          /* Tab breaks */
    MWSPut("L016","9;Airline\tTime\tGate");
}

```

## APPLE CONFIDENTIAL

DoFlight()

```
/******
 * This routine first creates an EXCEL file, then generates data
 * that is then written to the file. The file name is 'Schedule'.
 *****/
{
    int    error = false;

    MWSPut("F001","Schedule;XCEL;TEXT");    /* Create new file */
    MWSPut("F004","1;Schedule");            /* Open file data fork */

    WriteFlight();

    MWSPut("F006","1");                    /* Close file data fork */
    return(error);
}
```

WriteFlight()

```
/******
 * This routine writes 'hard-coded' data to a file.
 *****/
{
    int    i;

    MWSPut("F010","1;0;F;;");              /* Write data to the file */

    for (i=0; i<9; ++i)
        MWSPut("F011",FlightStr[i]);
    MWSPut("F012","");
    MWSPut("F016","1;0");                  /* Set file fork mark back to zero */

    MWSPut("L010","9");                    /* Send data to the list window */
    for (i=0; i<9; ++i)
        MWSPut("L011",FlightStr[i]);
    MWSPut("L012","");
}
```

## APPLE CONFIDENTIAL

ShowFlight()

```
/*
 * This routine shows the Flight Schedule window.
 */
{
    MWSPut("W003", "9");
}
```

DisposeFlight()

```
/*
 * This routine disposes of the Flight Schedule window.
 */
{
    MWSPut("W002", "9");
}
```

## APPLE CONFIDENTIAL

DoReservations()

```
/******
 *   This routine sets up the Reservations dialog box.   *
 *****/
{
    MWSPut("T001","3");          /* Cleans Up Previous Passenger Info */
    MWSPut("T010","3;Please Enter Special Meal Instructions ");
    MWSPut("T010","3;(close this window when done): ");
    MWSPut("D014","3;2;1");
    MWSPut("D014","3;3;0");
    MWSPut("D014","3;4;0");

    MWSPut("M009"," 1;1;1");

    if (DlgRes)
        MWSPut("D006","4;");      /* Show the hidden dialog */
    else {
        DlgRes = true;            /* Create the dialog */
        MWSPut("C002","4");
        MWSPut("D001","4;F;Bear Cal Reservation System;16;F;30,50,482,335");
        MWSPut("D009","4; B; 380,10,440,30; OK; F; T; 1");
        MWSPut("D009","4; B; 380,40,440,60; Cancel; T; F; 0");
        MWSPut("D009","4; T; 10,10,110,26; Passenger:");
        MWSPut("D009","4; E; 115,10,360,26; F; 1; T;");
        MWSPut("D009","4; T; 10,35,110,51; Destination:");
        MWSPut("D009","4; E; 115,35,360,51; F; 0; T; A; [");
        MWSPut("D009","4; T; 10,60,110,76; Depart:");
        MWSPut("D009","4; E; 115,60,185,76; F; 0; T;");
        MWSPut("D009","4; T; 205,60,285,76; Return:");
        MWSPut("D009","4; E; 290,60,360,76; F; 0; T;");
        MWSPut("D009","4; L; 12,85;440,85;");
        MWSPut("D009","4; T; 10,95,110,111; Airline:");
        MWSPut("D009","4; E; 115,95,330,111; F; 0; T;");
        MWSPut("D009","4; T; 10,120,110,136; Time:");
        MWSPut("D009","4; T; 1345,1120,1400,1136; Hidden:");
        MWSPut("D009","4; E; 115,120,330,136; F; 0; T;");
        MWSPut("D009","4; E; 1405,1120,1440,1136; F; 0; T;");
        MWSPut("D009","4; T; 10,145,110,161; Gate:");
        MWSPut("D009","4; E; 115,145,330,161; F; 0; T;");
        MWSPut("D009","4; T; 1235,1145,1295,1161; hidden:");
        MWSPut("D009","4; E; 1300,1145,1440,1161; F; 0; T;");
        MWSPut("D009","4; L; 12,170;440,170;");
        MWSPut("D009","4; T; 370,180,440,196; Payment:");
        MWSPut("D009","4; P; 375,240;101;F;F;1");
        MWSPut("D009","4; P; 375,200;100;F;F;1");
        MWSPut("D009","4; L; 360,175;360,275;");
    }
}
```



# APPLE CONFIDENTIAL

```
MWSPut("D009","4; T; 10,180,110,196; Incidentals:");
MWSPut("D009","4; C; 15,200,110,216; Smoking; T; T; 2; 0; 0;");
MWSPut("D009","4; C; 15,220,110,236; Dinner; T; T; 2; 0; 0;");
MWSPut("D009","4; C; 15,240,110,256; Movie; T; F; 2; 0; 0;");
MWSPut("D009","4; C; 15,260,110,276; Rental Car; T; T; 2; 0; 0;");
```

```
MWSPut("D009","4; T; 130,180,230,196; Class:");
MWSPut("D009","4; R; 135,200,230,216; First; T; F; 3; 0; 0;");
MWSPut("D009","4; R; 135,220,230,236; Business; T; F; 3; 0; 0;");
MWSPut("D009","4; R; 135,240,230,256; Coach; T; F; 3; 0; 1;");
```

```
MWSPut("D009","4; T; 250,180,350,196; Seating:");
MWSPut("D009","4; R; 255,200,350,216; Window; T; F; 4; 0; 0;");
MWSPut("D009","4; R; 255,220,350,236; Center; T; F; 4; 0; 1;");
MWSPut("D009","4; R; 255,240,350,256; Aisle; T; F; 4; 0; 0;");
```

```
MWSPut("D007","4");
MWSPut("C001","");
```

```
}
OnPayment = false;
```

ProcCancel()

```
/*
 * This routine is called if the user selects 'Cancel' from the
 * Reservations dialog box. It disposes of the Reservations dialog
 * box and enables the Reservations menu item.
 */
```

```
{
    MWSPut("D008","4");
    MWSPut("M008","0;1;1");
}
```

## APPLE CONFIDENTIAL

ProcOK()

```
/******
 * This routine is called if the user clicks 'OK' in the
 * Reservations dialog box. It checks to see if payment was made
 * the processes the payment.
 *****/
{
    char    EvtClass;
    int     EvtID;
    char    EvtParms[256];
    int     TempLoop = true;

    if (OnPayment) {
        SendToLog();          /* Enter passenger info into log */
        MWSPut("D008","4");    /* Hide Reservation dialog */
        MWSPut("M008","0;1;1"); /* Enable Bear Cal reservation item */
    } else {
        MWSPut("D001","9;T;;1;T;50,65,462,275");
        MWSPut("D009","9; P; 10,10;101;T;F;1");
        MWSPut("D009","9; P; 343,10;100;T;F;1");
        MWSPut("D009","9; B; 340,170,400,190; Cancel; T; F; 0");
        MWSPut("D009","9;T;10,80,402,140;You have not purchased your tickets yet.");
    }
}
```

# APPLE CONFIDENTIAL

```

while (TempLoop) {
    MWSGet(&EvtClass, &EvtID, EvtParms);
    if (EvtParms[0] == '9') {
        if (EvtParms[2] == '1') {          /* Generic Credit Card
                                           selected */

            if (EvtParms[6] == '1') {
                SendToLog();
                ProcGCC();
                MWSPut("D005", "9");
                MWSPut("D008", "4");
                MWSPut("M008", "0;1;1");
                SecondPayment = true;
            }
        }
        if (EvtParms[2] == '2') {          /* Super Card selected */
            if (EvtParms[6] == '1') {
                SendToLog();
                ProcSC();
                MWSPut("D005", "9");
                MWSPut("D008", "4");
                MWSPut("M008", "0;1;1");
                SecondPayment = true;
            }
        }
        if (EvtParms[2] == '3') {          /* Cancel selected */
            MWSPut("D005", "9");
            MWSPut("D008", "4");
            MWSPut("M008", "0;1;1");
        }
        TempLoop = false;
    }
}
}
}

```

ProcSC()

```

/*****
*   This routine is called if the user selects the Super Card
*   payment option. It displays a dialog box and copies the
*   Passenger name from the Reservations dialog box into the 'Name'
*   field of the Super Card dialog box.
*****/
{
    MWSPut("D002", "5;T");          /* Get Super Card dialog */
    WaitForResponse();              /* Pass over response */
    CopyName(5);                    /* Copy name into SC dialog */
}

```

## APPLE CONFIDENTIAL

ProcGCC()

```
/* *****
 * This routine is called if the user selects the Generic Credit
 * Card payment option. It displays a dialog box and copies the
 * Passenger name from the Reservations dialog box into the 'Name'
 * field of the GCC dialog box.
 * *****/
{
    MWSPut("D002","6;T");          /* Get GCC dialog */
    WaitForResponse();             /* Pass over response */
    CopyName(6);                   /* Copy name into GCC dialog */
}
```

AnswerSC()

```
/* *****
 * This routine is called when the user has entered the information
 * required in the Super Card dialog box. After it checks the
 * payment, the routine displays an authorization number and
 * thanks the user for flying Bear Cal airlines.
 * *****/
{
    MWSPut("D009","5;T;10,160,300,176;Authorization Granted (78342)");
    sleep(1);
    MWSPut("D005","5");

    if (SecondPayment)
        MWSPut("M008","0;1;1");

    OnPayment = true;
    SecondPayment = false;
}
```

# APPLE CONFIDENTIAL

AnswerGCC()

```

/*****
*   This routine is called when the user has entered information      *
*   required in the GCC dialog box. After it checks the payment, the  *
*   routine displays an authorization number and thanks the user for  *
*   flying Bear Cal airlines.                                         *
*****/
{
    MWSPut("D009","6;T;10,160,300,176;Authorization Granted  (3459)");
    sleep(1);
    MWSPut("D005","6");

    if (SecondPayment)
        MWSPut("M008","0;1;1");

    OnPayment = true;
    SecondPayment = false;
}

```

ProcSmoke()

```

/*****
*   This routine is called when the user turns on the 'Smoking'      *
*   checkbox. The routine displays an alert.                          *
*****/
{
    MWSPut("A002","F;WARNING: Bear Cal does not allow smoking on flights.");
}

```

ProcNotSmoke()

```

/*****
*   This routine is called when the user turns off the 'Smoking'     *
*   checkbox. The routine displays an alert.                          *
*****/
{
    MWSPut("A002","F;THANK YOU: We are pleased that you will not smoke.");
}

```

## APPLE CONFIDENTIAL

ProcRent ()

```
/******
 * This routine is called when the user turns on the 'Car Rental' *
 * checkbox. The routine displays a scrolling window of the cars *
 * available for rent. *
 *****/
(
  MWSPut("D001","8;F;Rental Options;0;T;100,50,412,300");
  MWSPut("D009","8;S;10,10,302,190;T;1;F");
  MWSPut("D009","8; B; 10,200,70,220; OK; F; F; 1");
  MWSPut("D009","8; B; 242,200,302,220; Cancel; T; F; 0");

  MWSPut("D016","8;1;Economy - Ford Festiva");
  MWSPut("D016","8;1;Economy - Nissan Sentra");
  MWSPut("D016","8;1;Economy - Chevrolet Sprint");

  MWSPut("D016","8;1;Compact - Ford Escort");
  MWSPut("D016","8;1;Compact - Toyota Corolla");
  MWSPut("D016","8;1;Compact - Mazda 323");

  MWSPut("D016","8;1;MidSize - Ford Taurus");
  MWSPut("D016","8;1;MidSize - Mazda 626");
  MWSPut("D016","8;1;MidSize - Chevrolet Cavalier");

  MWSPut("D016","8;1;Luxury - Lincoln Continental");
  MWSPut("D016","8;1;Luxury - Acura Legend");
  MWSPut("D016","8;1;Luxury - Cadillac Seville");

  MWSPut("D016","8;1;Sport - Ford Mustang GT");
  MWSPut("D016","8;1;Sport - Nissan 300 ZX");
  MWSPut("D016","8;1;Sport - Chevrolet Corvette");

  MWSPut("D016","8;1;Exotic - Lamborghini Contach");
  MWSPut("D016","8;1;Exotic - Ferrari Modial");
  MWSPut("D016","8;1;Exotic - Porsche 911 SC");
)
```

## APPLE CONFIDENTIAL

ProcNoRent()

```
/*
 * This routine is called when the user turns off the 'Car Rental'
 * checkbox. The routine displays an alert.
 */
{
    MWSPut("A002","F;Sorry that you have decided to cancel your car rental.");
}
```

ProcRentOK()

```
/*
 * This routine is called when the user clicks the 'OK' button
 * of the Car Rental dialog box.
 */
{
    MWSPut("D005","8");
}
```

ProcRentCancel()

```
/*
 * This routine is called when the user clicks the 'Cancel'
 * button in Car Rental dialog box.
 */
{
    MWSPut("D005","8");
    MWSPut("D014","4;31;0");
}
```

## APPLE CONFIDENTIAL

ProcDinner()

```

/*****
 * This routine is called when the user turns on the 'Dinner'
 * checkbox. It presents a dialog box that allows the user to
 * choose a meal.
 *****/
{
    int      TempLoop = true, EvtID;
    char      EvtClass, EvtParms[256];

    MWSPut("D007","3");          /* Make Meal Selection Dialog visible */
    while (TempLoop) {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'D') && (EvtID == 257)) {
            MWSPut("D008","3");
            switch(EvtParms[2]) {
                case '4': SpecialMeal();
                        break;
                case '5': TempLoop = false;
                        break;
                case '6': TempLoop = false;
                        MWSPut("D014","4;29;0");
                        break;
            }
        }
    }
}

```



# APPLE CONFIDENTIAL

ProcNoDinner()

```

/*****
*   This routine is called when the user turns off the 'Dinner'
*   checkbox. It presents a dialog box that allows the user to
*   cancel or change dinner.
*****/

{
    int      TempLoop = true, EvtID;
    char      EvtClass, EvtParms[256];

    MWSPut("D001","0;T;Dinnermod;3;F;166,122,356,242");
    MWSPut("D009","0;T;20,10,180,60;Do You Wish To Change or Cancel your
meal?");
    MWSPut("D009","0;B;20,75,75,95;Change;T;F;1");
    MWSPut("D009","0;B;105,75,160,95;Cancel;T;F;1");
    MWSPut("D007","0");
    while (TempLoop == true) {
        MWSPut("&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'D') && (EvtID == 257)) {
            TempLoop = false;
            MWSPut("D005","0");
            if (EvtParms[2] == '2') {
                MWSPut("D014","4;29;1");
                ProcDinner();
            }
        }
    }
}

```

## APPLE CONFIDENTIAL

SpecialMeal()

```
/******
 * This routine displays the Special Meal Text Window so that
 * special meal instructions can be entered.
 *****/

{
    int    TempLoop = true, EvtID;
    char    EvtClass, EvtParms[256];

    MWSPut("W003", "3");
    while (TempLoop) {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'W') && (EvtID == 257)) {
            MWSPut("W005", "3");
            MWSPut("D007", "3");
            MWSPut("M003", "0");
            TempLoop = false;
        }
    }
}
```

ShowLog()

```
/******
 * This routine displays the Passenger Log window.
 *****/

{
    MWSPut("W003", "7");
}
```

HideLog()

```
/******
 * This routine hides the Passenger Log window.
 *****/

{
    MWSPut("W005", "7");
}
```

# APPLE CONFIDENTIAL

SendToLog()

```

/*****
*   This routine adds Passenger Data to the Passenger Log.   *
*****/

{
    int    TempLoop = true, length;
    int    i;
    char    EvtClass;
    int    EvtID;
    char    EvtParms[256];
    char    AllStr[512];
    char    Count = 0;

    for(i = 0; i < 128; i++)
        EvtParms[i] = 0;
    for(i = 0; i < 256; i++)
        AllStr[i] = 0;

    AllStr[0] = '7';
    AllStr[1] = ',';
    Count += 2;

    MWSPut("D010", "4;4");
    while (TempLoop) {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'D') && (EvtID == 258)) {
            if (EvtParms[0] == '4') {
                if (EvtParms[4] == 'E') {
                    length = strlen(&EvtParms[6]);
                    strncpy(&AllStr[Count], &EvtParms[6], (length - 1));
                    Count += (length - 1);
                    AllStr[Count] = '\t';
                    Count++;
                    TempLoop = false;
                }
            }
        }
    }
}

```

## APPLE CONFIDENTIAL

```
MWSPut("D010","4;6");
TempLoop = true;
while (TempLoop) {
    MWSGet(&EvtClass, &EvtID, EvtParms);
    if ((EvtClass == 'D') && (EvtID == 258)) {
        if (EvtParms[0] == '4') {
            if (EvtParms[4] == 'E') {
                length = strlen(&EvtParms[6]);
                strncpy(&AllStr[Count], &EvtParms[6], (length - 1));
                Count += (length - 1);
                AllStr[Count] = '\t';
                Count++;
                TempLoop = false;
            }
        }
    }
}

MWSPut("D010","4;8");
TempLoop = true;
while (TempLoop) {
    MWSGet(&EvtClass, &EvtID, EvtParms);
    if ((EvtClass == 'D') && (EvtID == 258)) {
        if (EvtParms[0] == '4') {
            if (EvtParms[4] == 'E') {
                length = strlen(&EvtParms[6]);
                strncpy(&AllStr[Count], &EvtParms[6], (length - 1));
                Count += (length - 1);
                AllStr[Count] = '\0d';
                TempLoop = false;
            }
        }
    }
}

MWSPut("L009",AllStr);
}
```

# APPLE CONFIDENTIAL

RemoveFromLog (RecNum)

```

char    *RecNum;

/*****
 *   This routine removes the record RecNum from the Passenger Log
 *   after the user double-clicks on the record in the list window.
 *****/

(
char    AllStr[15], EvtClass, EvtParms[256];
int     EvtID, TempLoop = true;

MWSPut("D002", "2;T");
WaitForResponse();
MWSPut("A001", "");
AllStr[0] = '7';
AllStr[1] = ';';
while (TempLoop == true) {
    MWSGet(&EvtClass, &EvtID, EvtParms);
    if ((EvtClass == 'D') && (EvtID == 257)) {
        TempLoop = false;
        if (EvtParms[2] == '2') {
            strcpy(&AllStr[2], RecNum);
            MWSPut("L001", AllStr);
        } else {
            AllStr[2] = 'F';
            AllStr[3] = ';';
            strcpy(&AllStr[4], RecNum);
            MWSPut("L002", AllStr);
        }
    }
}
MWSPut("D005", "2");
}

```

# APPLE CONFIDENTIAL

CopyName(InInt)

```

    int    InInt;

    /*****
    *   This routine reads the passenger name from the Bear Cal dialog
    *   box and writes the name into the 'Name' item of the appropriate
    *   dialog box.
    *****/

    (
        int    TempLoop = true, length;
        char    EvtClass;
        int    EvtID;
        char    EvtParms[256];
        char    AllStr[256];
        int    Count = 0;
        int    i;

        for(i = 0; i < 128; i++)
            EvtParms[i] = 0;
        for(i = 0; i < 128; i++)
            AllStr[i] = 0;

        if(InInt == 5)
            AllStr[0] = '5';
        else if(InInt == 6)
            AllStr[0] = '6';

        AllStr[1] = ';';
        AllStr[2] = '8';
        AllStr[3] = ';';

        Count += 4;
        MWSPut("D010", "4;4");
        while (TempLoop) {
            MWSGet(&EvtClass, &EvtID, EvtParms);
            if ((EvtClass == 'D') && (EvtID == 258)) {
                if (EvtParms[0] == '4') {
                    if (EvtParms[4] == 'E') {
                        length = strlen(&EvtParms[6]);
                        strncpy(&AllStr[Count], &EvtParms[6], (length - 1));
                        TempLoop = false;
                    }
                }
            }
        }
        MWSPut("D011", AllStr);
    )

```

# APPLE CONFIDENTIAL

WaitForResponse()

```

/*****
 * This routine waits for an event from MWS, then ignores it if it
 * is not a Dialog Director, List Director, Menu Director, Process
 * Director, or Text Director event.
 *****/

{
    int    TempLoop = true;
    char    EvtClass;
    int     EvtID;
    char    EvtParms[256];

    while (TempLoop) {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'D') || (EvtClass == 'M') ||
            (EvtClass == 'L') || (EvtClass == 'T') || (EvtClass == 'P'))
            TempLoop = false;
    }
}

```

Quit()

```

/*****
 * This routine presents an alert that allows the user to cancel
 * a Quit command.
 *****/

{
    int    TempLoop = true, EvtID;
    char    EvtClass, EvtParms[256];

    MWSPut("A004", "T;Quit Bear Cal?");
    while (TempLoop == true) {
        MWSGet(&EvtClass, &EvtID, EvtParms);
        if ((EvtClass == 'A') && (EvtID == 256)) {
            TempLoop = false;
            if (EvtParms[0] == 'T')
                MWSPut("P003", "");
        }
    }
}

```

# APPLE CONFIDENTIAL

MWSGet(evtClass, evtID, evtMsg)

```

char    *evtClass;
int      evtID;
char     *evtMsg;

/*****
 *   This function gets a message from MacWorkStation and breaks
 *   it into its components. The protocol expected is:
 *       [      message begin character
 *       (data) message contents
 *       \n      message end character
 *   This conforms to the ID=2 transport-layer protocol.
 *****/
{
    char    begMsg;
    char    mwsMsg[512];

    gets(mwsMsg);
    sscanf(mwsMsg, "%c%c%3d", &begMsg, evtClass, evtID);
    if (strlen(mwsMsg) > 4)
        strcpy(evtMsg, &mwsMsg[4]);
    else
        *evtMsg = '\0';
    return 1;
}

```

MWSPut(cmdClass, cmdParms)

```

char    *cmdClass;
char     *cmdParms;

/*****
 *   This function sends a message to MacWorkStation.
 *   The protocol is:
 *       [      message begin character
 *       (data) message contents
 *       \n      message end character
 *   This conforms to the ID=2 Serial transport-layer protocol.
 *****/
{
    printf("%c%c%s%c", '[', cmdClass, cmdParms, '\n');
}

/* End of BearCal.c */

```



# Glossary

**alert box:** A box that appears on the screen when something has gone wrong or when something needs to be brought to the user's attention.

**alias:** A number assigned to an **object**. The number is then used by MacWorkStation and the client application to refer to the object.

**ASCII:** Acronym for *American Standard Code for Information Interchange*. The code, which is used for digital exchange of information between computers, printers, and so forth, assigns bit patterns to characters and tokens.

**Boolean operator:** A value that is either true or false.

**CCL:** See **Communication Command Language**.

**CCL script:** A resource stored in an MWS document that contains Communication Command Language commands. The script logs on to the **client application**. The script runs when the document is opened. The script may also contain log-off instructions.

**class:** Part of a MacWorkStation message, indicating the director to which the message belongs.

**client application:** An application that interacts with MacWorkStation.

**client computer:** The computer running the **client application**.

**cluster:** A group of **dialog box controls**. A control's cluster is set by a **parameter** in the command that creates the control.

**command:** A string of characters sent from a client application to MacWorkStation that instructs MWS to perform some function.

**Communication Command Language (CCL):**

The set of commands used in **CCL scripts** to log on and log off the client application.

**communication module:** Program code resource stored in MacWorkStation or in an MWS document that manages a communication protocol between MacWorkStation and the client computer.

**data fork:** The part of a Macintosh file that contains data. Macintosh files are composed of two parts, or forks: the **resource fork** and the data fork.

**director:** A set of MacWorkStation **commands** and **events** responsible for a group of related MacWorkStation functions, such as text display or file management. Directors call the Macintosh User Interface Toolbox **managers**.

**event:** A string of characters sent by MacWorkStation to the client application that informs the application about actions that the user has taken or about a MacWorkStation state.

**executable code module:** Program code resource stored in an MWS document that extend the capabilities of MWS.

**flag:** A MacWorkStation **parameter** that is a **Boolean operator**, which is either true or false.

**handshaking:** The process of establishing a connection between two computers in order to exchange data.

**identifier:** The part of a MacWorkStation message that describes the particular action to be performed or event that has occurred.

**integer:** A MacWorkStation **parameter type** composed of a sequence of characters, each of which has a value in the range of 0 to 9.

**keyword:** A string-type **parameter** with a fixed set of values used in commands.

**log on:** To establish a communication session between two computers. Log-on procedures may involve asking for identification, passwords, and so on. In MWS, log-on procedures are contained in **CCL scripts**.

**manager:** A set of Macintosh User Interface Toolbox routines responsible for a group of related functions, such as window or menu management.

**mark:** The current position in the sequence of numbered bytes of an MWS file, designating the next byte to be written or read.

**message:** A string of characters passed between MacWorkStation and the client application. Messages can be either **commands** or **events**.

**mode:** A mode is a part of an application that the user must formally enter and leave. A mode restricts the operations that can be performed while it is in effect.

**object:** Any element of the MacWorkStation display that can be treated as a single entity, such as a window, menu, or dialog box.

**parameter, parameter type:** (1) Part of a control message (**command** or **event**) that specifies the action a **director** is to take. (2) A setting that may assume more than one value.

**resource:** A formatted description of a user interface element in a Macintosh application. A resource can describe almost anything, including icons, menus, text strings, and the program code. Resources are stored in the **resource fork** of a file and loaded into memory as needed.

**resource fork:** The part of a Macintosh file that contains **resources**. Macintosh files are composed of a resource fork and a **data fork**.

**RGB monitor:** A type of color monitor that receives separate signals for the three colors: red, green, and blue.

**server computer:** A Macintosh computer running MWS and in communication with the **client application on the client computer**.

**string:** A MacWorkStation **parameter** type composed of a sequence of up to 255 characters.