Macintosh®

Introduction to Macintosh Programming Environments

Beta Draft

Mitchell Gass
Developer Technical Publications
March 20, 1989

Apple Confidential

20525 Mariani Avenue
Cupertino, CA 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

Contents

Figures and tables

Preface

PROGRAMMING ENVIRONMENTS ARE COLLECTIONS of tools and other facilities for developing computer software. This manual provides the information you need to choose a programming environment for the Apple® Macintosh® family of computers. It answers these important questions:

- How is developing software for the Macintosh family different from developing software for other computers?
- What are the advantages and disadvantages of available software development environments for the Macintosh family? What is it like to use each development environment?
- Where can I find the information I need to develop Macintosh software?

Introduction to Macintosh Programming Environments is written for anyone interested in developing software for the Macintosh family of computers, including

- professional software developers
- managers responsible for software development projects
- computer enthusiasts

It assumes that readers have programmed at least one other computer and that they are familiar with the Macintosh desktop interface. If you haven't yet used a Macintosh computer, you should become familiar with one or more Macintosh applications before reading this booklet. ■

Chapter 1 Programming for the Macintosh Family

SOFTWARE FOR THE MACINTOSH FAMILY is fundamentally different from software for other computers. This chapter describes these differences—a standardized user interface, event-driven programs, the User Interface Toolbox, and resources—and explains how they benefit software developers.

A standardized user interface

The Macintosh user interface is the single most important reason for the success of the Macintosh family. It provides a set easy-to-learn, intuitive techniques that make using the Macintosh a pleasure rather than a chore. For example, users needn't memorize commands; instead, they simply pull down menus to see the commands that are available. A variety of **direct manipulation** techniques, such as moving a file by dragging the file's icon, let users see the results of their actions.

A key advantage of the Macintosh user interface is its consistency: Once users are familiar with one Macintosh application, they can use the same skills in almost any other application. As a result, learning time and frustration are dramatically decreased.

The standard user-interface elements and their correct use in programs are described in an extremely useful book, *Human Interface Guidelines: The Apple Desktop Interface*. This book distills years of research and user testing into practical guidelines for program design. By following the guidelines, you will ensure that your programs are both easy and efficient to use.

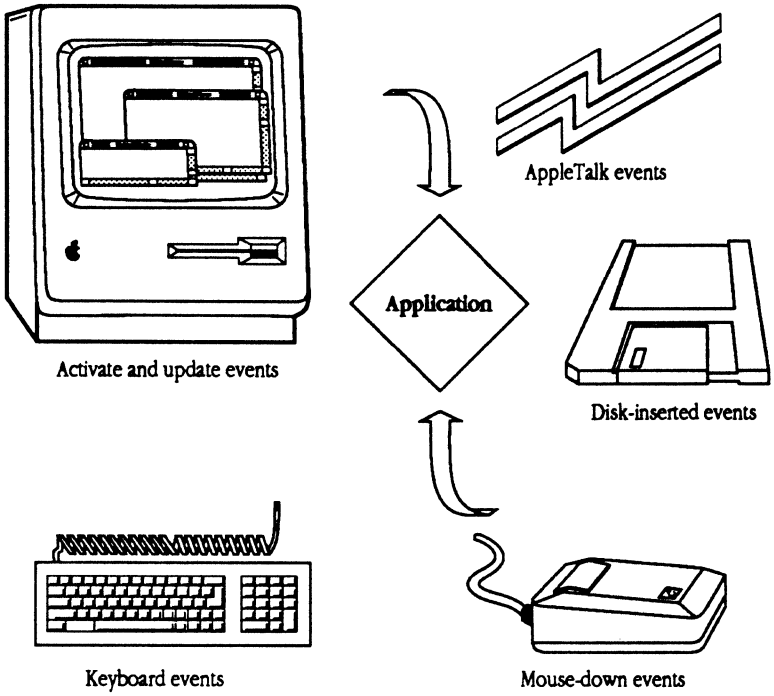
Event-driven programs

In all well-written Macintosh programs, the user is in charge. The user guides the interaction with the computer—using the available menus, windows, dialog boxes, and other controls—rather than the other way around. This responsiveness to the user distinguishes Macintosh programs from conventional programs.

The actions a user performs to control the Macintosh—such as a press of the mouse button or a key on the keyboard—are known as **events.** To respond to these events, Macintosh programs must be **event-driven:** they need to constantly check for incoming events, determine the significance of each event, and respond as necessary.

Events include keyboard events, mouse events, and disk-insertion events. Activity on a communications network can also generate events. These events are shown in Figure 1-1.

■ **Figure 1-1** Types of events



At the core of each Macintosh program is an **event loop** that handles incoming events. Figure 1-2 illustrates what happens in a typical event loop.

■ **Figure 1-2** Flow of control in an event loop

```
make standard initialization calls
put up window
### begin main event loop ###
wait for events
      key-down event?
            Command key down?
                  yes:  do a command
                  no:  accept typing from user
      mouse-down event?
            where is the mouse?
                  is it the front window?
                        no:  bring it to the front
                        yes:  user is selecting something
                  in a desk accessory window?
                        pass event to the Desk Manager
                  in the menu bar?
                        call Menu Manager, then do command
                  in the window's title bar?
                        call Window Manager to drag the window
                  in the window's size box?
                        call Window Manager to resize the window
                  in a control (button, scroll bar, ...)
                        call Control Manager to find out which
                          control
                        then act on that control
      activate event? (a new window is being activated)
            enable some menu items, disable others, etc.
      update event? (part of a window needs to be redrawn)
            redraw the window
repeat until the user chooses "Quit" from the File menu
```

Because the events a Macintosh program must respond to are almost always the same, an event loop is an easily reused piece of code. This means that you never need to write a Macintosh program from scratch; if you don't have a good example of the kind of program you're writing, you begin with an already-written event loop.

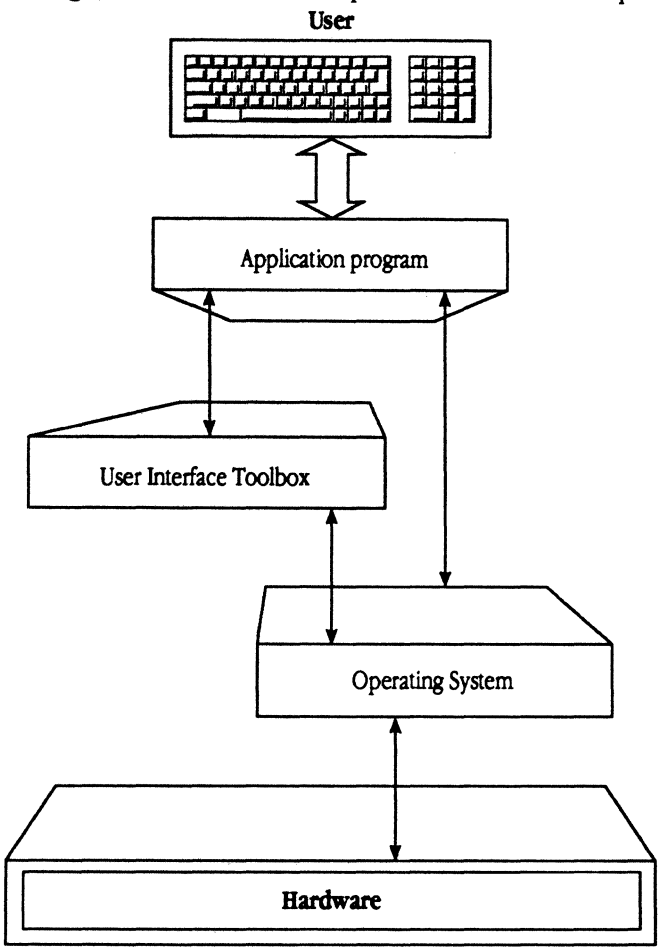
The User Interface Toolbox

The designers of the Macintosh realized that the easiest way to get programmers to use the standard Macintosh user interface was to give them all the code they need to implement it. This code is known as the User Interface Toolbox. The Toolbox provides routines for creating menus, windows, dialog boxes, and all of the other standard elements of the Macintosh user interface. It also provides support for handling events. Thanks to the Toolbox, it's much easier to implement the standard Macintosh interface than to create a non-standard interface of your own.

Because Toolbox routines are highly optimized, using them also saves execution time over equivalent routines you'd write yourself. Another advantage of using the Toolbox is that the routines are guaranteed to work in future versions of the Macintosh. Thus, using Toolbox routines whenever possible helps to ensure the future compatibility of your programs.

Most of the Toolbox routines are stored in the Macintosh ROM, while the remaining Toolbox routines are loaded into RAM when the Macintosh starts up. The routines can be called from any programming language that provides the appropriate interfaces. In turn, the Toolbox routines call lower-level routines in the Macintosh Operating System to perform specific tasks. This relationship is illustrated in Figure 1-3.

■ **Figure 1-3** Relationship of Toolbox calls to Operating System calls



Resources

A **resource** is a chunk of data that is used by a Macintosh program. For example, the definitions for a program's menus, windows, and icons are normally stored as resources. *Any* information that is used by a program can be stored as a resource; in fact, even the pieces of code that make up a program are resources. You can thus think of a Macintosh program as a collection of resources.

The advantage of resources is that they allow the data used by a program to be stored separately from the program's code. This makes it possible to change a program's data without changing its code. For example, suppose you want to create a French version of a program that was originally written for English speakers. You open all the resources that contain text for the program, such as the text in the program's menu and dialog box resources, replace the text with French, and *voilà*, you've created a French version without ever touching the code. Moreover, easy-to-use tools like ResEdit, a Macintosh resource editor, let ordinary users change a program's resources. This gives users more control over the programs they use and lets non-programmers perform routine program maintenance.

There are many standard types of resources that you can use in your programs. These include the menu, window, icon, and text resources mentioned above. You can also create new resource types for data that is specific to a program.

Chapter 2 A Quick Look at Available Development Environments

THIS CHAPTER PROVIDES BRIEF DESCRIPTIONS of the most widely used Macintosh development environments. Read it to find one or more environments that are appropriate for your needs and then turn to the following chapters for more complete descriptions of those environments. ■

Choosing a development environment

The environments described in the following sections are arranged in order of ease of use. The easiest of these, HyperCard®, allows even a beginner to create useful programs in just a few hours. The most sophisticated environment, the Macintosh Programmer's Workshop (MPW™), is most appropriate for professional programmers who are already familiar with other full-featured development environments.

HyperCard

Of the software development environments for the Macintosh, HyperCard is by far the easiest to use. It is ideal for beginning programmers and programmers who need quick results. It can also be used to develop user-interface "front ends" for other applications, including applications that run on other computers. Avoid HyperCard if your application needs user-interface features that HyperCard doesn't provide, such as multiple windows and color displays, or if your application performs "number crunching" or other processor-intensive tasks.

MacWorkStation

MacWorkStation™ is used for applications in which a Macintosh is connected to another computer. It lets you build applications that combine the best features of the Macintosh, such as its forgiving user interface, with the capabilities provided by the remote computer. Moreover, you don't have to know how to program the Macintosh to use MacWorkStation. The program you write for the remote computer send messages to request Macintosh services. The MacWorkStation program running on the Macintosh receives these messages and makes the appropriate User Interface Toolbox calls to fulfill each request.

MacApp

MacApp® is an object-oriented application framework that simplifies the creation of full-featured Macintosh applications. Although it requires time to learn, this time is quickly repaid when you create your first application. It can only be used to create applications, however; to create other kinds of programs, such as desk accessories and device drivers, you must use another programming environment.

Other development environments

There are many other development environments for the Macintosh. Among them are LightspeedC and Lightspeed Pascal, two popular environments that you can use to create any kind of Macintosh program. Macintosh implementations of the LISP, Prolog, Smalltalk, and FORTH programming languages include their own integrated development environments. A/UX®, a version of the AT&T UNIX operating system, is also available.

MPW

The Macintosh Programmer's Workshop (MPW) is the most powerful and versatile development environment currently available for the Macintosh. You can use MPW to write any kind of program for the Macintosh, including desk accessories, device drivers, programming tools, and standalone code resources. In addition, MPW provides a variety of useful features, such as shell scripts and support for multilingual programs, that aren't available in other Macintosh development environments.

Chapter 3 **HyperCard**

OF THE SOFTWARE DEVELOPMENT ENVIRONMENTS FOR THE MACINTOSH, HyperCard is by far the easiest to use. It is ideal for beginning programmers and programmers who need quick results. It can also be used to develop user-interface "front ends" for other applications, including applications that run on other computers. Avoid HyperCard if your application needs user-interface features not provided by HyperCard, such as multiple windows and color displays, or if it performs "number crunching" or other processor-intensive tasks. ■

About HyperCard

Bill Atkinson, the creator of HyperCard, describes his creation as "an attempt to bridge the gap between the priesthood of programmers and the Macintosh mouse clickers." By any measure, the attempt is a wild success. In a matter of hours, almost anyone can learn to create useful HyperCard programs.

HyperCard is an application for storing and retrieving information. In a HyperCard program, known as a **stack,** any piece of information—which can be text, graphics, or sound—can be linked to any other piece of information. It's also a kind of software "erector set" that lets you assemble **objects,** the building blocks of HyperCard, to create programs. If you haven't seen HyperCard in action, be sure to visit a Macintosh owner or dealer and get a demonstration of its power and versatility.

Included with HyperCard is an easy-to-use programming language known as **HyperTalk™.** In this chapter, you'll learn about the special advantages of HyperCard and HyperTalk as a programming environment and get a taste of what it's like to develop HyperCard stacks.

Advantages of HyperCard

If you've used HyperCard, you know how useful it can be for end users. It can also be a very productive environment for programmers.

Easy to learn

HyperTalk, the programming language included with HyperCard, is one of the easiest programming languages to learn. Learning how to create a HyperCard stack requires only a fraction of the time needed to learn any of the other programming environments described in this booklet.

Speeds development

HyperCard handles nearly all of the work that goes into creating the user interface for a stack. This means that you can concentrate on what you want in a stack rather than the details of how to create the windows and other features, as you would with a conventional programming language.

Because of the ease of HyperCard development, HyperCard can also be used to develop quick prototypes for other programs.

Provides a rich environment

HyperCard provides easy access to the text, graphics, and sound capabilities of the Macintosh family. Some of the best features of the Macintosh are now available to anyone who learns HyperTalk.

Universal availability

HyperCard is now "standard equipment" for Macintosh computers. A copy of HyperCard is bundled with all new Macintosh computers now being shipped. In addition, users with older Macintosh computers can obtain HyperCard for a very modest price. Because of this, the potential audience for HyperCard stacks is very large.

Extensibility

You can use conventional programming languages to add functionality to HyperCard stacks. A stack can call external commands (known as XCMDs) and external functions (known as XFCNs) which contain executable code written in other programming languages. This approach gives you the best of both: the power and flexibility of conventional programming together with the simplicity and speed of HyperCard development.

The extensibility of HyperCard also allows it to be used as the user-interface "front end" for other programs.

Disadvantages of HyperCard

There is, of course, a price to pay for the ease of HyperCard development: HyperTalk is not as powerful or versatile as conventional programming languages.

Can only be used to develop stacks

HyperTalk is not a general-purpose programming tool. To develop Macintosh applications and other kinds of programs, you must use a conventional programming language.

Fewer features than other programming languages

HyperTalk is easy to use because it is a simple language. It does not provide the wealth of data types, data structures, and control constructs available in most programming languages.

No direct access to the Toolbox

HyperCard stacks cannot make direct calls to Macintosh Operating System and User Interface Toolbox routines. To use these routines, you must create standalone code resources known as external commands (XCMDs) and external functions (XFCNs) that make the necessary calls. These resources, which you write using a conventional programming environment such as MPW, can be called by stacks in the same way that they call HyperCard's built-in commands and functions.

Using HyperCard

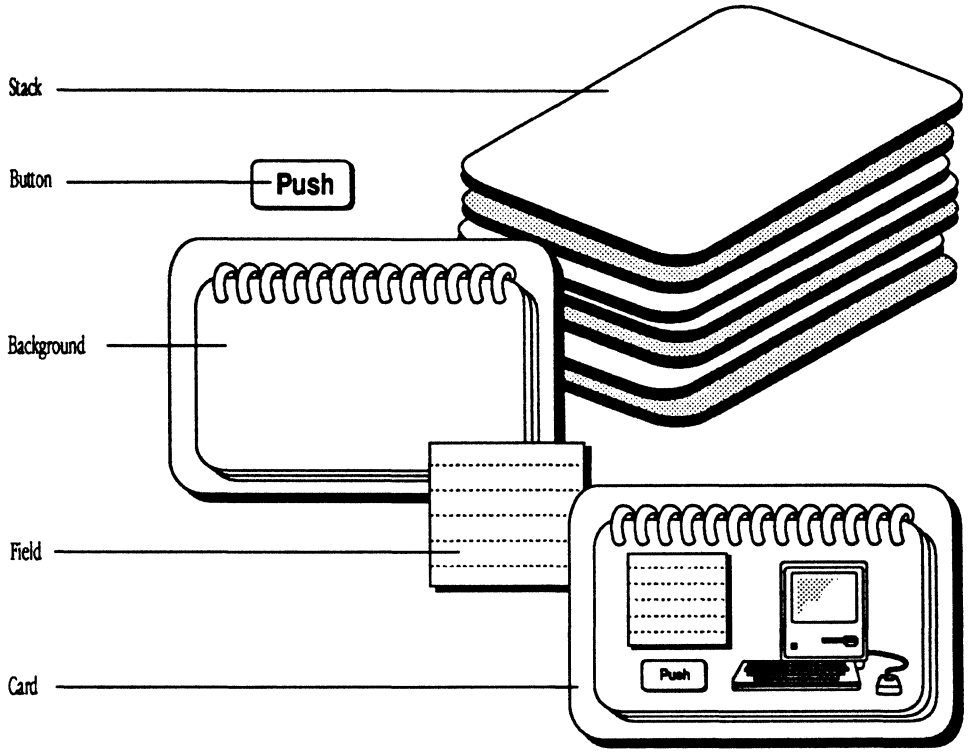
Because the power of HyperCard is so accessible, creating HyperCard stacks is less like programming and more like the work done by writers and artists. You spend less time wrestling with the computer and more time thinking about how to organize information and how to present it creatively.

HyperCard is a flexible tool that lets you develop stacks in a number of ways. Beginners can modify existing stacks or assemble stacks from pieces of other stacks. Large stacks, on the other hand, lend themselves to collaborative work by teams of specialists, such as stack designers, artists, writers, and HyperTalk programmers. The following sections describe the two tasks that are common to all stack development: creating **objects** and writing **scripts.**

Creating objects

A HyperTalk stack is composed of **objects.** These objects include the stack itself, the cards that make up the stack, and the backgrounds, fields, and buttons for cards. These objects are illustrated in Figure 3-1:

■ **Figure 3-1** HyperCard objects



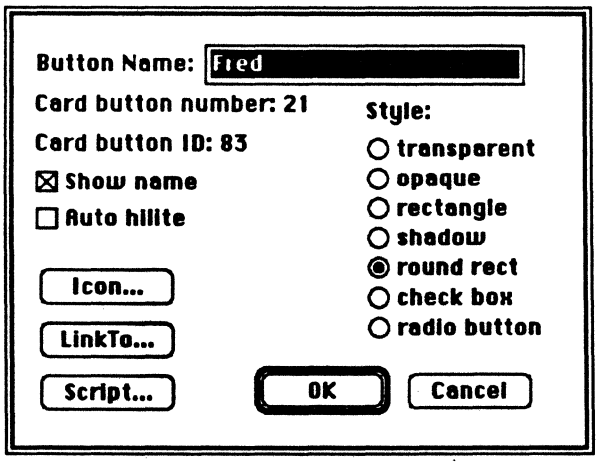
The first step in creating your stack is to specify the objects that make up the stack. You create the necessary artwork for backgrounds, collect sounds to be used in the stack, and determine the contents and layout of the cards.

To create objects, you first enable HyperCard's programming facilities by clicking the Scripting button on the User Preferences card in the Home stack. You can then create an object by selecting one of the "New" menu items from the Objects menu:

■ **Figure 3-2** The Objects menu

If you select an object and choose the appropriate Info item from the Objects menu, you can see and change the attributes of the object. For example, here is the dialog box that appears when you choose the Button Info menu item:

■ **Figure 3-3** Button Info dialog box



Writing scripts

Once you've created the objects for a stack, the next task in creating your stack is specifying the relationships between objects. For example, you decide that clicking a particular button causes the next card in the stack to appear and that clicking a button in that card plays a reggae version of "Yankee Doodle Dandy." These relationships are sometimes called a **navigational model;** they specify how the user moves through the stack. For a stack of any size, careful planning is essential: you need to have a clear picture of the entire stack and its operation before you implement it.

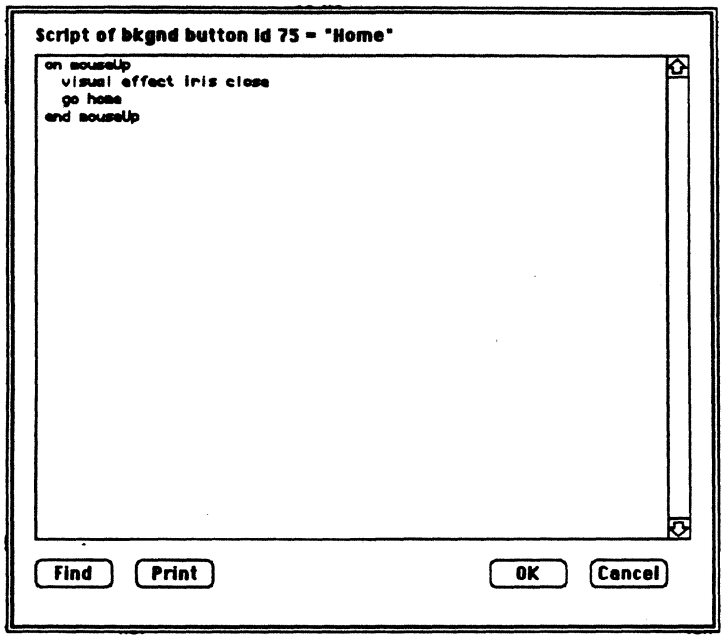
To define the navigational model, you write scripts. Scripts are sets of **handlers** which specify what happens when **messages** are received by an object. In turn, handlers are composed of HyperTalk statements. For example, suppose a user points to a HyperCard button and clicks the mouse button. When the user releases the mouse button, HyperCard sends a `mouseUp` message to the button. The script for the button contains the following handler for the `mouseUp` message:

```
on mouseUp
      go to next card
end mouseUp
```

This handler displays the next card in the stack.

To see or change the script for an object, you select the object, choose the appropriate Info menu item from the Objects menu, and then select the Script button. This invokes the **script editor:**

■ **Figure 3-4** Script editor box



Chapter 4 **MacWorkStation**

MACWORKSTATION LETS YOU PUT A MACINTOSH "FACE" on an application running on another computer. Use it to build applications that combine the best features of the Macintosh with the capabilities provided by mainframes and minicomputers.

About MacWorkStation

MacWorkStation is one of a family of products from Apple that lets Macintosh computers work with other computers, including mainframes, minicomputers, and other personal computers. MacWorkStation allows a Macintosh user and a program running on a remote computer—known as a **host application**—to interact through the Macintosh user interface. It provides support for all the familiar features of the Macintosh user interface, including pull-down menus, windows, and dialog boxes. Thus, you can create a host application that, to a Macintosh user, looks and works like any other Macintosh application.

Like MacApp, MacWorkStation is not a programming environment in itself, but is instead a set of tools for developing applications. It includes a Macintosh application that controls the user interface for the application, communication modules that support a variety of data-communication standards, and protocols for communication between the host application and the Macintosh application.

Advantages of MacWorkStation

MacWorkStation is exceptionally useful for many kinds of applications. Its main advantages are described in the following sections.

Provides the full power of the Macintosh

MacWorkStation delivers the most important feature of the Macintosh: the Macintosh user interface. Host applications are no longer limited to the line-oriented interface provided by terminal emulation programs. Instead, they can include pull-down menus, windows, and all the other user-interface features that Macintosh users expect. MacWorkStation can also handle such tasks as printing from the Macintosh and local file management. Applications developed with MacWorkStation can also include code that performs local processing of data on the Macintosh, including code that takes advantage of Macintosh graphics, sound, and other features.

Eliminates Macintosh programming

The MacWorkStation application that runs on the Macintosh handles all the standard features that MacWorkStation provides. You don't need to write any Macintosh code to take advantage of these features, nor do you have to understand the details of the Macintosh Operating System or User Interface Toolbox routines.

Doesn't require host processing time

All of the processing required for MacWorkStation features, including the management of the Macintosh user interface, is performed on the Macintosh. The host application is freed to perform other tasks.

Speeds application development

Because MacWorkStation takes care of so many of the tasks associated with application development, you can develop sizable applications in weeks rather than months.

Is extensible

Applications developed with MacWorkStation can include executable code modules that run on the Macintosh. These modules—which are similar in purpose to HyperCard XCMDs—provide access to any features of the Macintosh. They can perform local processing of data, transform incoming data to graphic form for display on the Macintosh, or gain access to any of the Macintosh Operating System or User Interface Toolbox routines.

You can also create custom communication modules for MacWorkStation that can handle new networks and protocols.

Disadvantages of MacWorkStation

Although you can use the underlying power of the Macintosh Operating System and User Interface Toolbox routines with MacWorkStation, you are forced to use a conventional programming language to create routines that are called by MacWorkStation. By itself, MacWorkStation does not provide access to Macintosh Operating System or User Interface Toolbox routines.

Using MacWorkStation

Every MacWorkStation application has two components: a **client** application that runs on a remote computer and the MacWorkStation **server** application that runs on a Macintosh. The server serves the client by making the features of the Macintosh user interface (and, optionally, local processing of data) available to the client. The following sections describe how you create these two components and get them to work together.

Writing the client application

The client application has three responsibilities:

- acting on **messages** it receives from MacWorkStation
- processing data on the host computer
- sending messages to MacWorkStation to request the Macintosh services it needs

For example, suppose you're using MacWorkStation to let Macintosh users retrieve information from a remote database. When the user chooses the Retrieve Info menu item, MacWorkStation sends a message to the client to let it know that the menu item has been chosen. The client application interprets the message and acts on it by reading the required data from the database. To display the information, the client sends MacWorkStation a message to open a window on the Macintosh Desktop. When MacWorkStation receives the message, it calls the necessary Macintosh Operating System and User Interface Toolbox routines to open the window.

Selecting a communication method

You can use any of a number of networks to connect the client and server, including serial, SNA, AppleTalk®, and Ethernet. To communicate over the network, you specify a **communication module** to handle low-level data transport between the client and server. This module hides the details of session management from your application.

MacWorkStation provides three standard communication modules that work with a wide variety of computers. You can also write your own communication modules.

Writing additional code

More elaborate applications can include Macintosh code modules that perform additional processing. To create these executable code modules, you can use any Macintosh development that can generate standalone code resources.

Writing the log-on and log-off scripts

You next write scripts that allow the user to automatically log on to and log off of the host computer. You write these scripts in **Communications Command Language (CCL),** a special-purpose language provided with MacWorkStation.

Creating the MacWorkStation document

The last step in writing a MacWorkStation application is creating a **MacWorkStation document.** Like other Macintosh documents, this document appears as an icon on the Desktop. Users can run the finished application by opening its document.

The document contains

- the log-on and log-off scripts
- the communication module used by the application
- the resources used by the application, such as menu, window, and dialog box resources

The document can also contain any executable code modules used by the application.

Chapter 5 **MacApp**

MACAPP SIMPLIFIES THE CREATION OF FULL-FEATURED MACINTOSH APPLICATIONS. Although it requires time to learn, this time is quickly repaid when you create your first application. It can only be used to create applications, however; to create other kinds of programs, such as desk accessories and device drivers, you must use another programming environment. ■

About MacApp

MacApp, the Expandable Macintosh Application, is designed to make creating a Macintosh application as efficient as possible. It is a generic application, built using object-oriented techniques, that contains all the user-interface features of a desirable Macintosh application: menus, windows, dialog boxes, and more. When you create an application with MacApp, nearly all the work required for the user interface is already complete. This frees you to create the parts of your application that are unique.

MacApp is not a programming environment in itself. It must be used with MPW or another programming environment that supports it.

Advantages of MacApp

Of all the development tools available to Macintosh programmers, MacApp may be the most valuable. The following sections explain why.

Reduces coding required for applications

By providing all the code needed for standard application elements, such as windows, menus, and dialog boxes, MacApp eliminates nearly all of the coding required to develop the user interface for an application. The closer an application conforms to the ideal user interface that is built into MacApp, the less coding is required.

Speeds application development

Once you understand MacApp and object-oriented programming, you can develop significant applications in weeks rather than months.

Provides excellent code

All the code used in MacApp is highly optimized and comparable to the best code used in commercial applications.

Provides benefits of object-oriented programming

Object-oriented programming techniques give MacApp much of its power. MacApp is built using Object Pascal—better known as MPW Pascal—which adds object-oriented extensions to standard Pascal. To create your own application with MacApp, you must use an object-oriented language such as Object Pascal or TML Pascal.

Object-oriented programming offers many important benefits. These include better management of program complexity (due, in part, to the way objects encapsulate data and procedures associated with the data), an increase in the ability to reuse code (due to the property of inheritance in object-oriented systems), and increased productivity.

Helps applications conform to human-interface guidelines

As provided, the MacApp application conforms closely to the Apple human-interface standard. In general, the less work you do to change this interface (which includes the user-interface features required by nearly all applications), the more closely the resulting application will conform to the standard.

Assures compatibility with future systems

The features included in MacApp are guaranteed to work with all future versions of Macintosh and A/UX system software. Support for MultiFinder and for other important Macintosh features is already built in. Using MacApp is the best insurance that future changes and enhancements to the system software will not affect your application.

Includes source code

MacApp includes complete source code for all its components. The code serves both to document applications built with MacApp and as a learning tool for MacApp programmers.

Includes necessary makefiles

To make creating your application as easy as possible, MacApp includes the MPW makefiles needed to generate object code for your application. This reduces the burden of learning MPW that is required of MacApp users.

Disadvantages of MacApp

For all its advantages, MacApp does have some significant drawbacks. These drawbacks are described in the following sections.

Requires time to learn

Learning MacApp requires a considerable investment of time. You must gain a knowledge of MacApp, an object-oriented programming language, and the MPW development environment that underlies MacApp. This time is typically repaid—with interest—when developing your first application with MacApp.

Can only be used to develop applications

MacApp can only be used to develop applications. To develop other kinds of programs, such as desk accessories and device drivers, you must use another programming environment.

Limited choice of languages

The only high-level programming languages you can currently use with MacApp are MPW Pascal and TML Pascal II. For code where speed is essential, you can use MPW assembly language together with the object-oriented macros it includes.

Requires MPW

At present, the only software development environment that supports MacApp is MPW. All of the programming languages listed in the previous section, including TML Pascal II, require MPW.

Using MacApp

If you've used an object-oriented programming language, such as Smalltalk or C++, the fundamentals of MacApp will already be familiar to you. The building blocks for MacApp applications are **objects,** packages that contain both data and routines, known as **methods,** for operating on the data. You define a **class** of objects by specifying the data type for each of the data fields, known as **instance variables,** and defining each of the class's methods. An individual object is an **instance** of an object class; its instance variables and methods are the same as those for all other objects that belong to the same class.

Much of the power of object-oriented programming derives from the properties of objects. Objects, by combining data and procedures, let you build complex programs out of small, easily managed units. Another property, known as **inheritance,** allows the fields and methods of an object class to be derived from other class. For example, suppose that there is an object class called Animal. Another class, Mammal, has the same instance variables and methods as the Animal class but changes the definitions for a few of the methods. In this way, defining the Mammal class, which is a **subclass** of Animal, is very simple: you say that it is inherited from Animal and redefine only those methods that need to be different from Animal. Moreover, inheritance makes it easy to reuse code: when you define a method for a class, the same method is available in any subclasses of the class.

Starting with a sample application

A number of sample programs are provided with MacApp. The simplest of these, Nothing, includes the standard features of a Macintosh application, such as a window and menus, but does nothing more than display a word in the window. It is an ideal foundation for your own programs. Other sample programs show how you can use MacApp to create fully functional applications.

Building a sample application

There are five source files for the each sample application:

- The main program (such as MNothing.p)
- The Object Pascal interface for the object classes (such as UNothing.p)
- The implementation of the object classes (such as MNothing.p)
- The resource definition file (such as UNothing.r)
- An MPW **make** file that contains instructions for building the application (such as UNothing.make)

To build the application, you run an MPW script known as MABuild. For example, you can build the Nothing application by entering

```
MABuild Nothing
```

The MABuild script, which is included with MacApp, creates a finished application. After you build an application, you can run it by simply entering its name in an MPW Shell window and pressing Enter.

Creating your own application

To create your own program with MacApp, you first copy the five files for the Nothing application and change the identifiers containing Nothing (such as the filename UNothing.p) to equivalent identifiers containing your program name. You now have all the files for your new application.

You next initialize the Toolbox, which makes User Interface Toolbox calls available to your application. You also initialize any of the additional services available from MacApp, such as printing, that are needed by your application.

You then create your first object, the **application object,** which dispatches the events that are received by the application. The application passes some of these events to MacApp for handling, while others are handled by the application object itself. To create the application object, you begin with a class provided by MacApp called TApplication and create a subclass of that class. This subclass provides the exact functionality required by your application. In the subclass, you **override** one of TApplication's methods, replacing the method with one you write yourself. You then create an instance of the class which becomes a component of the application. Finally, you call one of the object's methods to initialize the object.

Adding functionality

To add functionality to a MacApp application, you often perform the steps described for creating an application object: you begin with a predefined class, create a subclass that meets the specific needs of your application, and then create an object which is an instance of the new subclass. For example, you normally create one or more document objects which contain the data produced by the application. You use the TDocument class provided by MacApp as the basis for your own subclass and then create an instance of the new subclass.

Among the other objects you add for a typical application are objects for

- windows
- scroll bars for windows
- the contents of windows
- menu commands
- mouse tracking commands

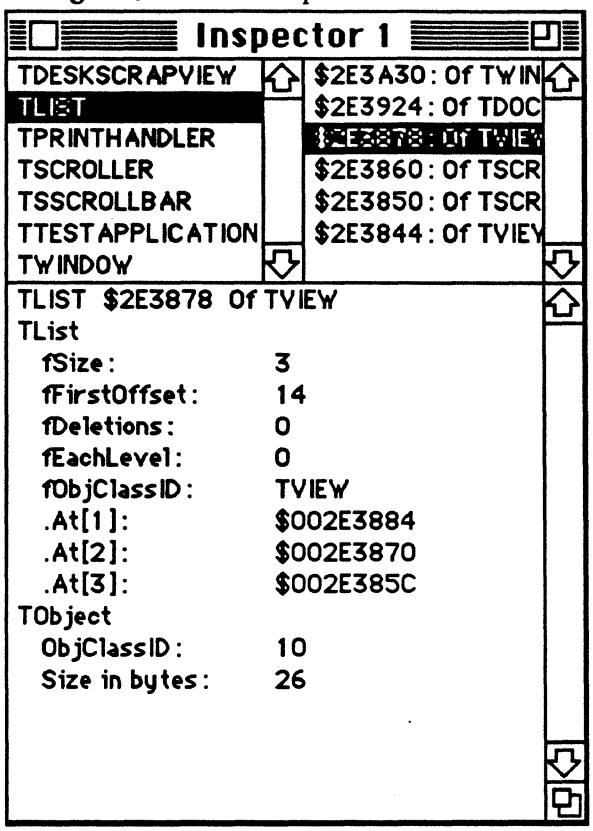
Adding resources

You add other components of your application, such as menus, menu items, and icons, by adding definitions to your application's resource definition file. When you build the application, the MABuild script invokes Rez, the MPW resource compiler, to create the resources.

Debugging your application

To debug your application, you use the **Inspector** tool that is provided with MacApp. A sample Inspector window is shown in Figure 5-1.

■ **Figure 5-1** An Inspector window



In the upper left corner of the Inspector window is a **class list,** a list of all the classes in your application for which there are instances. Clicking on an entry in the class list displays an **instance list** in the upper-right corner of the window. This list contains the name and address of each instance of the selected class. Clicking on an instance in this list displays the values of the instance's fields (shown here in the bottom portion of the window), including the values of any fields that it has inherited.

Chapter 6 The Macintosh Programmer's Workshop

THE MACINTOSH PROGRAMMER'S WORKSHOP (MPW) is the most powerful and versatile development environment currently available for the Macintosh. You can use MPW to write any kind of program for the Macintosh, including desk accessories, device drivers, programming tools, and standalone code resources. In addition, MPW provides a variety of useful features, such as shell scripts and support for multilingual programs, that aren't available in other Macintosh development environments. ■

About MPW

The Macintosh Programmer's Workshop (MPW) is the most powerful and versatile of the software development environments for the Macintosh. Written by Apple Computer for its own software development, MPW is now available to all developers. This chapter provides a brief overview of MPW, describes its advantages and disadvantages, and gives you a taste of using MPW and its many features.

MPW is a direct descendant of earlier software-development environments for the Macintosh, in particular the Lisa Workshop. It is a robust, full-featured development environment designed to meet the needs of professional software developers.

In its design and underlying philosophy, MPW owes much to the UNIX development environment. MPW and UNIX share such features as a command shell, support for aliases and shell variables, I/O redirection, pipes, shell scripts, command histories, and regular expressions. In addition, both environments can be extended to suit the needs of those using them.

Advantages of MPW

MPW offers many important advantages for both professional developers and enthusiasts. These advantages are described in the following sections.

Provides a full-featured development environment

MPW provides an extremely wide range of tools and facilities to help you develop software. It includes a project management system (Projector), a linker and librarian, a Make utility to streamline the rebuilding of programs generated from multiple files, a resource compiler (Rez) and a decompiler (DeRez), and performance-measurement tools. Also provided with MPW are three programming utilities that can be used with any programming environment: a resource editor (ResEdit™), a debugger for high-level languages (SADE), and a debugger for assembly language (MacsBug).

A variety of programming languages is available for MPW. Languages available from Apple include Pascal, C, and assembly language. All three of these languages include complete interface libraries for the Macintosh Operating System and User Interface Toolbox. Other vendors provide their own versions of C and Pascal as well as languages such as Modula-2 and FORTRAN.

Programs developed with MPW be "multilingual." For example, you can write the bulk of a program in a high-level language like C or Pascal, write any routines where speed is critical in assembly language, and then link the components together to create a finished program.

MPW also provides a variety of features designed to make programming easier and more productive. For example, you can include segmentation controls anywhere within a file, allowing you to organize your code according to its logical structure rather than the desired segmentation.

Provides an integrated development environment

The MPW Shell integrates the features of a command interpreter and a full-screen editor. All the components of MPW can be used within the MPW Shell environment. In addition, you can quickly move from the Shell to general-purpose programming utilities such as ResEdit and SADE by using MultiFinder™.

Supports large development projects

MPW was written with large development projects in mind. It can easily handle projects with teams of programmers, hundreds of files, and many thousands of lines of source code. The current version includes Projector, an integrated project-management system that maintains revision histories and other information needed to keep big projects on track.

Creates any kind of program

With MPW, you can create any kind of programs for the Macintosh, including applications, desk accessories, MPW tools, device drivers, the external commands (XCMDs) and external functions (XFCNs) used by HyperCard, and other standalone code resources. If it will run on the Macintosh, you can create it with MPW.

Reflects latest development at Apple

All Macintosh system software is developed with MPW. This means that MPW is kept up-to-date and reflects the latest product development at Apple.

Lets you create your own environment

You can easily tailor MPW to fit your needs and style of working. For example, you can add menus and menu items (complete with keyboard equivalents), write command scripts to perform common tasks, alter existing scripts (including the startup and user startup scripts used to configure the MPW environment), and create aliases for existing commands.

The ability to create Shell scripts—a feature missing from other development environments—can be extremely useful for repetitive tasks and for processing large files. Scripting allows you to combine pieces of code to create useful utilities. For example, the following three-line script adds a menu item to the MPW Find menu, opens a dialog box when the menu item is chosen, searches all your source files for the text string you enter in the dialog box, displays any matches in the MPW Worksheet window, and disposes of any error messages:

```
addmenu Find 'Search sources for…' ∂
      '(Search /"`Request "Look for what in source files?"`"/ ∂
      "{mpw}mysources:"≈ >> "{Worksheet}") ≥ dev:null '
```

You can also use MPW to build your own programming tools. These tools, known as **MPW tools,** are special programs that run within the MPW environment. Most of the MPW Shell commands are, in fact, MPW tools; when you add a new tool, it becomes part of the MPW environment.

Because the MPW Shell takes care of the user interface for MPW tools, developing an MPW tool is much easier and faster than creating an equivalent Macintosh application. Moreover, tools created for UNIX or other development environments can typically be ported to MPW with very little modification. You can also choose from a variety of MPW tools, including compilers and debuggers, that are offered by other companies.

Supports object-oriented programming

Object-oriented programming is a relatively new approach to programming, first popularized by the Smalltalk development environment, that uses **objects**—data structures with associated procedures—as the building blocks for programs. Object-oriented programming offers important benefits, including better management of program complexity, an increase in the ability to reuse code, and increased productivity. MPW Pascal and assembly language include extensions that support object-oriented programming.

Works with MacApp

MacApp (described in Chapter 5) is an object-oriented framework for creating new applications. It is an application shell, written in MPW Pascal, with all the standard features of a Macintosh application—windows, menus, dialog boxes, and more—built in. To create a new application, you remove any features you don't need, write the code needed to fill the windows, and handle any processing that is independent of the user interface. As a result, creating an application with MacApp typically takes a fraction of the time it would take without MacApp.

At present, you must have both MPW and either MPW Pascal or TML Pascal to use MacApp. MPW Pascal provides a number of optimizations that are included specifically for MacApp and object-oriented programs. In the future, other development systems are likely to include support for MacApp.

Disadvantages of MPW

MPW has a few important disadvantages, which are described in the following sections.

Complexity

Because of its wealth of features, MPW takes longer to learn than many other development environments. This extra investment in time is repaid, however, for those undertaking large or complex programming tasks. Moreover, MacApp users can ignore many of the details of MPW, such as the build process, that are taken care of automatically by MacApp.

Slow compiling and linking

Compiling and linking typically takes longer in MPW than in other development environments. This is due in part to optimization performed by MPW compilers (for such features as method-table lookup used in object-oriented programs) and to Linker optimizations such as the automatic removal of unused code.

Hardware requirements

To use all the features of MPW, you must have a Macintosh with at least two megabytes of memory and a hard disk. To use MPW together with SADE under MultiFinder, or to use MacApp with MPW, you need four megabytes of memory. In contrast, a number of other development environments can run well on a less fully configured Macintosh.

No choice of text editors

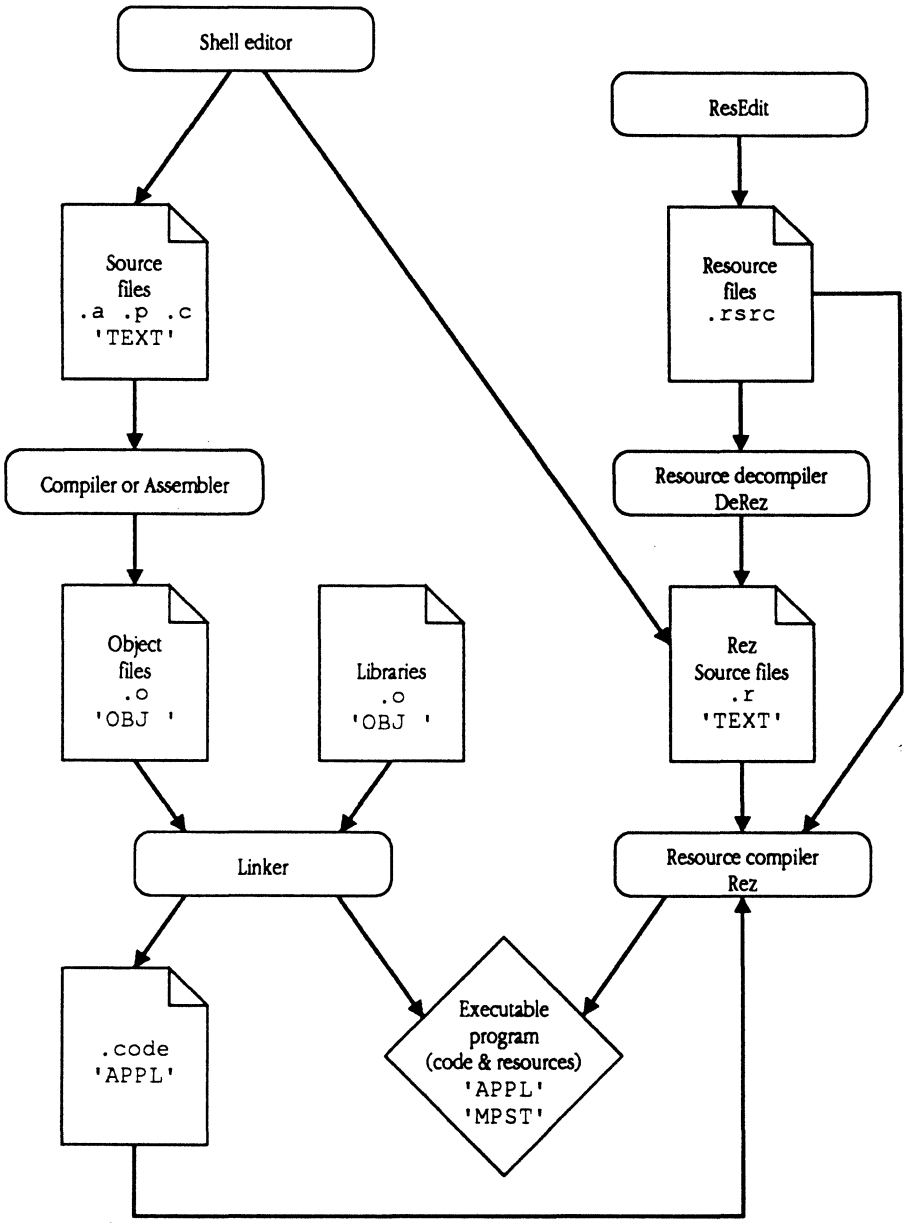
Because the MPW text editor is so tightly integrated with the rest of the MPW environment, it is difficult to use another text editor with which you may be more familiar. Moreover, the MPW text editor lacks many of the features—such as programmability—that are provided by Emacs and other widely used editors. The MPW editor is both flexible and easy to learn, however, and for most programmers its advantages outweigh these disadvantages.

Using MPW

The following sections provide a brief description of working with MPW. While not exhaustive, these sections do give you a taste of using MPW and what it can do for you. As you read, you'll also learn about many of the components of MPW.

Figure 6-1 illustrates the steps involved in developing software with MPW.

■ **Figure 6-1** Developing software with MPW



Starting with a sample program

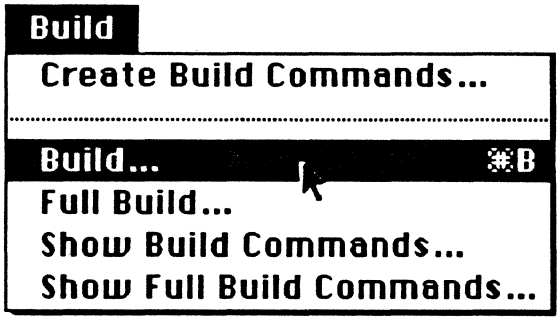
When you create a program with MPW, you never need to start from scratch. Provided with MPW are sample applications, desk accessories, and MPW tools written in C, Pascal, and assembly language. You simply copy an appropriate sample and use it as a model for your own program.

Building the sample program

Before you modify one of the sample programs, you'll want to run the program and have a look at what it does. To do this, you first **build** the program by compiling and linking all of its components.

One of the advantages of MPW is the ability to automate the building of programs. For example, you can build any of the sample programs provided with MPW in two easy steps. First, you select the directory that contains the sample program's files by choosing Set Directory from the Directory menu. Once you've a specified the directory, you choose Build from the Build menu. This runs all the MPW tools, such as compilers and the linker, that are necessary to create the program:

- **Figure 6-2** The Build menu



To run the finished program, you simply type its name in any MPW Shell window and press Enter.

Expanding the program

You're now ready to expand and modify the sample program. To create new source files for the program, you use the mouse-based text editor that is integrated with the MPW Shell. The text editor includes the same cut, paste, search-and-replace, and undo features that are available in other Macintosh applications.

Because of tight integration of the editor with the command interpreter, you can use the editor to edit any text in any window. This is true even if the text is the output from a previous command. You can also select text in any window and execute it as a command by pressing Enter.

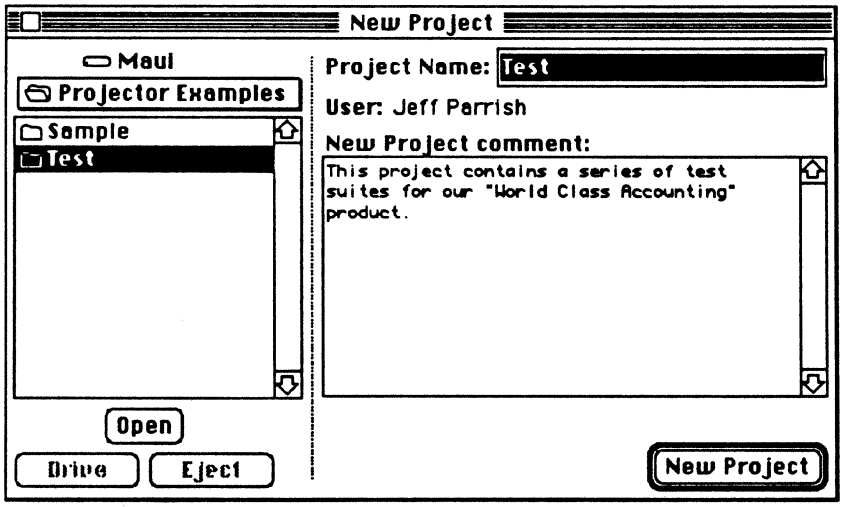
Resource description files are typically among the source files you create. These files are compiled with **Rez,** the MPW resource compiler. Rather than create all the resources you need from scratch, you can modify existing resources with **ResEdit,** the general-purpose resource editor that is provided with MPW. You can then decompile the resulting resources with **DeRez,** the MPW resource decompiler, to create resource descriptions that become part of your program.

Using Projector

You next use **Projector,** an integrated project-management system that maintains revision histories and other information about your programs. Projector allows you record the changes you make to your program's source files and to keep records of alternate versions of your files. If there are several programmers working on the project, you can also use Projector to control access to files so that only one person can modify a file at the same time.

To begin using Projector, you create a new **project.** A project contains all the information about a program's files and revisions that is maintained by Projector. When you choose New Project from the Project menu, a New Project window like the one illustrated in Figure 6-3 appears:

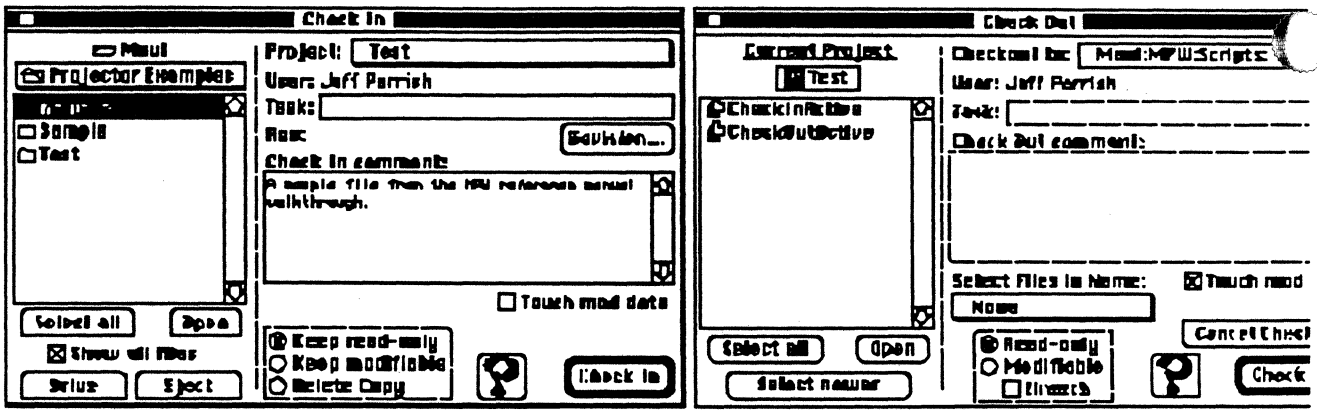
■ **Figure 6-3** A New Project window



You then type in the project's name and and a description of the project.

Once you've created a project, you check out the files you want to work with and check them back in when you're done. Figure 6-4 illustrates the windows you use to enter information when checking out and checking in files.

■ **Figure 6-4** The Check Out and Check In windows

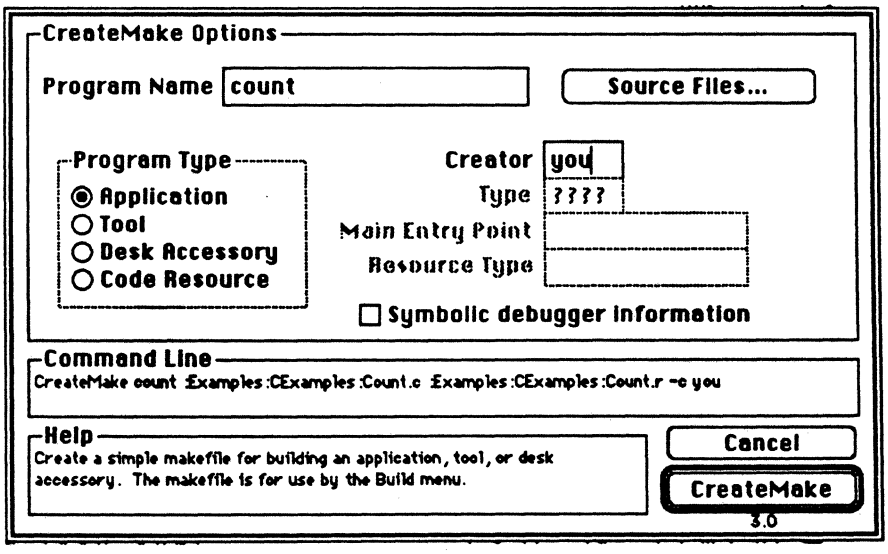


Projector gives you a great deal of control over your projects. For example, you can create experimental versions of files and later merge the successful experiments into the the project. Projector also keeps all the information needed to recreate any previous version of the project.

Adding build commands

Whenever you add you new files for a program, you need to add additional build commands for the program. You do this by choosing Add Build Commands from the Build menu. When you, the CreateMake dialog box illustrated in Figure 6-5 appears:

■ **Figure 6-5** The CreateMake dialog box



You type in the program's name (leaving off any suffixes such as ".a" or ".p") and select a radio button to indicate what kind of program it is. You then select the Source Files... button and specify the files that make up the program. When you're done, you can again build the program by choosing Build from the Build menu.

Debugging the program

Once you've built your program, you can debug it with one of the MPW debugging tools: MacsBug, an assembly-language debugger, or SADE (Symbolic Application Debugging Environment), a multiwindow symbolic debugger designed for use with high-level languages such as Pascal and C. SADE is particularly useful because it allows you to create your own debugging scripts. You can use these scripts to automate repetitive tasks and to create custom debugging tools.

Measuring the program's performance

When your program is working properly, you can check its performance with the performance-measuring tools provided with MPW. These tools, which include a set of measurement routines and a performance report generator, measure the amount of time a program spends executing any piece of code that you specify. You can then isolate heavily used or inefficient code and optimize it for speed.

Chapter 7 Other Development Environments, Languages, and Tools

THERE ARE MANY MORE SOFTWARE DEVELOPMENT ENVIRONMENTS, languages, and tools for the Macintosh than have been described so far. This chapter provides an overview of the many other products that are available. It also includes descriptions of the two most widely used programming languages for the Macintosh, LightspeedC and Lightspeed Pascal, and benchmarks that help you compare these and other popular languages. ■

Development Environments

There are now Macintosh implementations of many widely known development environments. These include the environments for

- LISP
- Smalltalk
- Prolog
- FORTH
- UNIX

These and other environments are described in *Apple II and Macintosh Development Software: A Guide to Languages and Tools,* a brochure produced by the Languages and Tools Product Marketing Group at Apple Computer. (For more information about this brochure, see Chapter 8.) The version of UNIX for the Macintosh, known as A/UX, is described in the next section.

A/UX

A/UX is Apple's version of the AT&T UNIX System V operating system. It is fully compatible with all System V software and includes many extensions from the Berkeley 4.2 BSD version of AT&T UNIX. A/UX runs on any member of the Macintosh II family with a paged memory-management unit (PMMU), an 80-megabyte or larger hard disk, and at least two megabytes of memory. At least four megabytes of memory are required for A/UX software development or for use with a network.

A/UX is an alternative to the usual Macintosh operating system that provides

- support for multiple users and multitasking
- a wide range of tools for software development and text processing
- a variety of command-line interfaces ("shells") and support the X Window windowing environment
- the ability to run Macintosh and UNIX programs in the same environment

Although you can't use A/UX to write standard Macintosh programs, programs you write for A/UX can call a subset of the Macintosh User Interface Toolbox routines. This lets you add some of the familiar features of Macintosh programs, such as windows and menus, to A/UX programs.

Languages

There is an extremely wide range of programming languages for the Macintosh. These include versions of the Smalltalk, LISP, Prolog, and FORTH languages mentioned earlier and several versions of the C and Pascal programming languages. Among the other languages are versions of

- FORTRAN
- COBOL
- BASIC
- Modula-2
- Logo
- APL
- Ada

Many of these languages (and all the languages listed in the next section) are described in *Apple II and Macintosh Development Software: A Guide to Languages and Tools*, a brochure produced by the Languages and Tools Product Marketing Group at Apple Computer. For more information about this brochure, see Chapter 8.

Languages for MPW

There are a number of programming languages that can be used with the Macintosh Programmer's Workshop (MPW). Available from Apple are versions of Pascal, C, and assembly language. Among the languages available from other vendors are

- Aztec C68k from Manx Software Systems
- Language Systems FORTRAN from Language Systems Corporation
- MacFortran/MPW from Absoft
- TML Modula-2 from TML Systems
- SemperSoft Modula-2 from Semper Software
- TML Pascal II from TML Systems

Tools

Just as with programming languages, the range of tools available for Macintosh software development is extremely wide. These tools include

- expert-system development tools
- application generators
- debuggers
- code libraries

Many of these tools are described in *Apple II and Macintosh Development Software: A Guide to Languages and Tools*, a brochure produced by the Languages and Tools Product Marketing Group at Apple Computer. For more information about this brochure, see Chapter 8.

Comparisons of popular programming languages

The following sections describe LightspeedC and Lightspeed Pascal and provide benchmarks for these and other widely used languages.

LightspeedC and Lightspeed Pascal

LightspeedC and Lightspeed Pascal are the most widely used versions of C and Pascal available for the Macintosh. Their popularity is due to several key advantages, which are described in the following sections.

Fast compiler and linker

Probably the most important advantage of LightspeedC and Lightspeed Pascal is their speed. Both compiling and linking are very fast, taking a few seconds or less when you've made only small changes.

Easy to learn

Both LightspeedC and Lightspeed Pascal are relatively easy to learn and use. The languages are particularly valuable for new Macintosh developers and developers working alone or in small groups.

Integrated environment

Both LightspeedC and Lightspeed Pascal include a compiler, a linker, an integrated text editor, a Make facility, project management capabilities, and a source-level debugger. They also include interfaces to all Macintosh Operating System and User Interface Toolbox routines.

LightspeedC includes the standard C library (with full source code) and interfaces to the Macintosh Operating System and Toolbox.

Project management

LightspeedC and Lightspeed Pascal keep all the files needed to build a program in a single unit known as a **project.** The necessary components for each kind of project (application, desk accessory, device driver, or code resource) are created automatically when the program is built. In addition, information about the dependencies between a project's files is maintained automatically.

Automatic Toolbox initialization

Before calling any User Interface Toolbox routines (described in Chapter 1), a program must initialize the Toolbox Managers that contain the routines. Both LightspeedC and Lightspeed Pascal initialize some of the more commonly used Toolbox managers. This saves you time and helps you avoid errors that result from initializing the managers in the wrong order.

Can be used to develop any kind of program

Like MPW, both LightspeedC and Lightspeed Pascal can be used to develop any kinds of Macintosh programs, including applications, desk accessories, device drivers, and stand-alone code resources.

Inexpensive

Because a complete development environment is included with LightspeedC and Lightspeed Pascal, the total cost of a useful development system is quite small.

Benchmarks

[To be provided]

Chapter 8 **Information for Software Developers**

THIS CHAPTER TELLS WHERE YOU CAN FIND INFORMATION you need to develop software for the Macintosh. It lists some of the most useful books, manuals, product descriptions, and information services currently available to developers. The list is far from complete; many other valuable books, publications, and services are also available.

Introductory information

Before developing software for the Macintosh, you should become familiar with the information in these three valuable books:

- *Technical Introduction to the Macintosh Family* (published by Addison-Wesley)
  An excellent introduction to the hardware and software design of the Macintosh family. It is the first book about Macintosh development you should read.
- *Human Interface Guidelines: The Apple Desktop Interface* (published by Addison-Wesley)
  Provides the information you'll need to design easy-to-use programs for the Macintosh.
- *Programmer's Introduction to the Macintosh Family* (published by Addison-Wesley)
  An introduction to Macintosh software for Macintosh software developers. It explains how the programs you'll write will work with the other software components of the Macintosh.

General references

The following books contain important reference information for Macintosh programmers:

- *Inside Macintosh,* Volumes I-V (published by Addison-Wesley)
  The essential guide to Macintosh System Software and User Interface Toolbox routines. A must for all Macintosh developers.
- *Macintosh Family Hardware Reference* (published by Addison-Wesley)
  The definitive guide to Macintosh hardware. Essential for software developers working on hardware-related projects.
- *Designing Cards and Drivers for Macintosh II and Macintosh SE* (published by Addison-Wesley)
  Essential for software developers working on card- or driver-related projects.
- *Inside Macintosh X-Ref* (published by Addison-Wesley)
  An index to all of the books listed above, including all five volumes of *Inside Macintosh*. Highly recommended.
- *Programmer's Guide to MultiFinder* (published by Addison-Wesley)
  Provides the information necessary to create applications that work with MultiFinder. Essential for anyone developing Macintosh applications.

- *Apple Numerics Manual* (published by Addison-Wesley)
  Provides information about SANE®, the Standard Apple Numeric Environment, and the data types and operations it provides. Necessary for developers whose programs use the numeric data types and numeric processing routines provided by SANE.
- *Macintosh Revealed: Volume 1, Unlocking the Toolbox,* 2nd edition
  Macintosh Revealed: Volume 2, Programming with the Toolbox, 2nd edition
  Both by Stephen Chernikoff (published by Hayden Books)
  The best currently available guides to programming the Macintosh family. They include many useful programming examples.
- *How to Write Macintosh Software,* 2nd edition, by Scott Knaster (published by Hayden Books)
  Includes useful discussions of memory management, debugging, and other topics of interest to Macintosh programmers.
- *Macintosh Programming Secrets* by Scott Knaster (published by Addison-Wesley)
  Provides tips for getting the most out of Macintosh programs, development tools, and hardware.
- *Macintosh Technical Notes*
  These provide addenda and revisions to Macintosh technical information as well as documentation for known bugs. They are sent to certified developers by the Macintosh Technical Support Group at Apple. Back issues are available from the Apple Programmer's and Developer's Association (APDA™). (For more information on APDA, see the "ADPA" section later in this chapter.). The notes are available in printed form and on disk.

MPW

Apple provides a number of reference manuals with MPW. The following manuals are included with all versions of MPW:

- *Macintosh Programmer's Workshop Reference,* which describes the MPW Shell, Shell commands, and other features of MPW
- *ResEdit Reference*
- *MacsBug Reference*
- *SADE Reference*

The following manuals are included with their respective languages:

- *Macintosh Programmer's Workshop C Reference*
- *Macintosh Programmer's Workshop Pascal Reference*
- *Macintosh Programmer's Workshop Assembler Reference*

There is also an excellent book that describes MPW and Macintosh programming:

- *Programming with Macintosh Programmer's Workshop* by Joel West (published by Bantam Books)

MacApp

Two MacApp manuals are currently available from Apple:

- *Introduction to Object-Oriented Programming with MacApp 2.0*
  Explains the principles of object-oriented programming as well as the design and structure of MacApp.
- *MacApp 2.0 Tutorial*
  A step-by-step guide to creating an application with MacApp.
- *MacApp 2.0 Cookbook*
  A how-to guide to the many features of MacApp.

All of these manuals are included with MacApp.

HyperCard

Apple produces two guides to HyperCard:

- *HyperCard User's Guide*
  Included with HyperCard.
- *HyperCard Script Language Guide: The HyperTalk Language* (published by Addison-Wesley)
  Describes the components of HyperTalk and how to use it to create HyperCard scripts.

There are many other excellent books about HyperCard. These include:

- *The Complete HyperCard Handbook,* 2nd Edition, by Danny Goodman (published by Bantam Books)
  A guide to all aspects of HyperCard, including scripting.
- *Danny Goodman's HyperCard Developer's Guide* by Danny Goodman (published by Bantam Books)
  Describes how develop HyperCard stacks.

- *HyperCard Power: Techniques and Scripts* by Carol Kaehler (published by Addison-Wesley)
  Provides useful tips for using HyperCard and for creating and modifying scripts.
- *XCMD's for HyperCard* by Gary Bond (published by MIS: Press)
  Explains how to write external commands (XCMDs) and external functions (XFCNs) for HyperCard.

MacWorkStation

Apple produces two manuals for MacWorkStation developers:

- *MacWorkStation Programmer's Guide* (available from APDA)
  A tutorial for developers using MacWorkStation.
- *MacWorkStation Programmer's Reference* (available from APDA)
  A complete reference to the components of MacWorkStation.

Guides to other development products

Apple II and Macintosh Development Software: A Guide to Languages and Tools
A brochure produced by Languages and Tools Product Marketing at Apple Computer that lists all programming languages, development tools, and programming libraries for the Macintosh that are available for sale. Copies of this brochure are available from [???where???]

A Guide to Apple's Multivendor Communication Products
A catalog produced by the Network and Communications Product Marketing Group at Apple Compute that lists products that facilitate communication between the Macintosh Family and larger computers. Copies of this brochure (Marketing Number M5116) are available from Apple Marketing Representatives or the Network and Communications Product Marketing Group.

APDA

The Apple Programmer's and Developer's Association (APDA) is an excellent source for development environments, tools, and reference books, including all of those described in this chapter. Membership includes an informative newsletter that is published regularly. For information, contact

APDA
Apple Computer, Inc.
20525 Mariani Avenue, Mailstop 33-G
Cupertino, CA 95014-6299
1-800-282-APDA or 1-800-282-2732
Fax: 408-562-3971
AppleLink: DEV.CHANNELS

Apple Developer Programs

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 Computer, Inc.
20525 Mariani Avenue, Mailstop 51-W
Cupertino, CA 95014-6299

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and Microsoft® Word. Proof pages were created on the Apple LaserWriter® II NTX. POSTSCRIPT®, the LaserWriter page-description language, was developed by Adobe Systems Incorporated. Some of the illustrations were created using Adobe Illustrator™.

Text and display type is ITC Garamond® (a downloadable font distributed by Adobe Systems). Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier, a fixed-width font.