Introduction Chapter 1

1

What You'll Learn

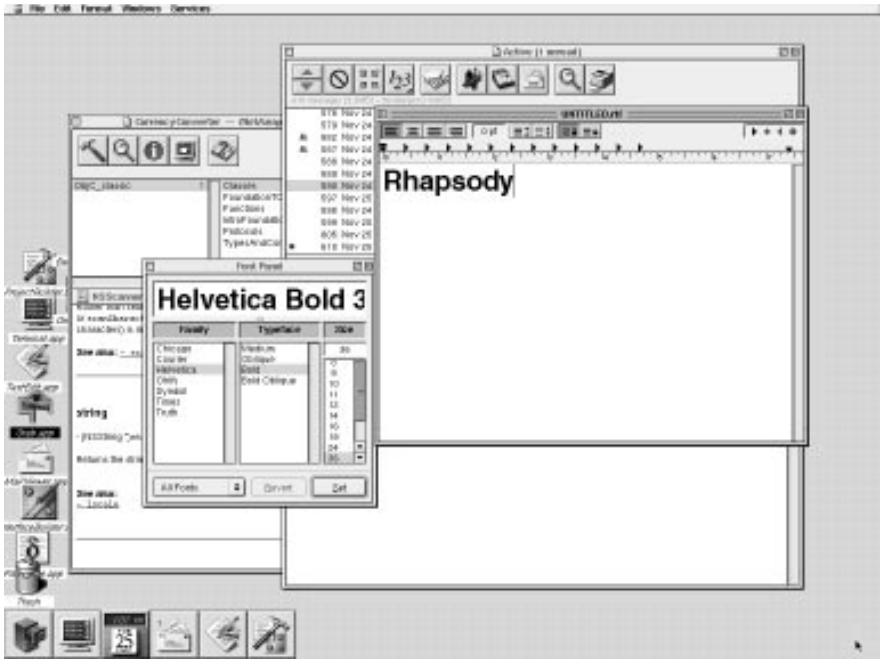
What OpenStep is

How OpenStep fits in with Rhapsody

Rhapsody technologies

Programming in Apple's development environment

Programming with objects



Chapter 1
Welcome to Rhapsody

Welcome to OpenStep.

OpenStep is a new way to make programs that run on Power PC Macintoshes. But OpenStep itself is not new. It is proven technology, and although it poses a learning curve for newcomers, once you learn it, application development will suddenly seem easier and quickened with potential.

This book eases your way into the experience of OpenStep programming. It guides you through the creation of several applications. It encourages you to explore, to "kick the tires." Along the way, it explains important concepts and paradigms, and it uncovers rich lodes in the tools and APIs.

When you've worked through this book, Apple's development environment will not only be less mysterious, but will be an environment that you'll want to program in.

This chapter presents a brief overview of OpenStep—the user experience and the developer experience—and places it in the larger context, which is the next-generation Macintosh operating system.

Imagine a Macintosh...

Imagine a Macintosh:

- That doesn't crash when an application crashes.
- That can draw a graph, send a fax, and play a movie, all at once.
- That seems much faster than any Windows system you've seen.
- That looks a lot like the Mac OS, only better.
- That features some of the most advanced software technology around.
- And that still runs all the Macintosh programs you know and love.

That Macintosh will soon be here. And you're invited to be a part of its genesis.

What is OpenStep?

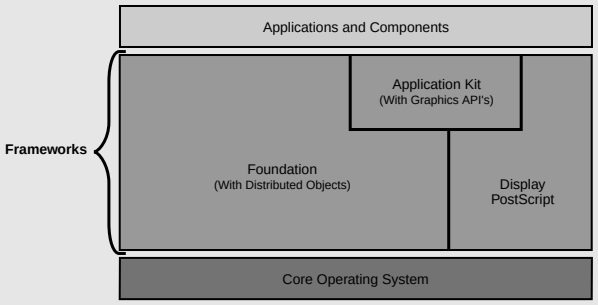
The new Macintosh operating system, code-named "Rhapsody," has actually been around a long time—almost as long as the Mac itself. That's because it is based on OpenStep, which, as NEXTSTEP, was introduced in 1987. Since then, OpenStep has evolved through many releases, has been adopted by many companies as their development and deployment environment of choice, and has received glowing reviews in the press. It is solid technology, based on a design that was years ahead of anything else and perfected year after year.

So OpenStep is well-regarded and battle-tested. But what is it? OpenStep is an integrated set of shared object libraries, or *frameworks*, plus a run time and a development environment that do three principal things:

- They insulate programs from the internal workings of the core operating system, mediating access to system resources and preventing programs from trashing one another's address space.
- They provide all (or almost all) the functionality that programs typically need.
- They bring the benefits of object-orientation to program development (see page 14).

OpenStep has three core object frameworks: Foundation, Application Kit, and Display PostScript.

Foundation Defines basic object characteristics, and implements mechanisms for object allocation, deallocation, introspection, and distribution. Foundation includes classes for common data types, such as strings, numbers, and collections. And it provides APIs for platform-independent system services, such as dates and times, multithreading support, task and process management, timers, file management, notification, and internationalization (based on the Unicode standard),



Application Kit Consists of classes that generate user-interface objects, that manage and process events, and that offer or assist in application services such as color and font management, printing, text manipulation, and cut-and-paste.

Display PostScript Provides APIs for direct PostScript drawing and image composition, as well as for low-level window, cursor, and event management. The Application Kit uses these APIs to draw the objects of the user interface. (New Graphics APIs will offer a higher level of abstraction for drawing operations.)

Both the Foundation and Display PostScript frameworks interact directly with the core operating system (Display PostScript with the windowing and imaging subsystem); applications can access operating-system services through the APIs of these frameworks.

The part of Rhapsody called the Yellow Box augments and enhances the core OpenStep frameworks with many other frameworks. See page 10 for details.

OpenStep and Mach

You can think of OpenStep as a layer of objects that acts as mediator and facilitator between programs and the core operating system. The stability, performance, and reliability of these programs therefore depend on whether the underlying core operating system has these characteristics.

The core operating system for Rhapsody is an enhanced version of Mach and BSD. Mach—the original foundation of NEXTSTEP—is a mature, robust kernel that provides low-level services such as memory management, tasking, synchronization, timing, and event messaging. These services form the basis of advanced operating-system capabilities: preemptive multitasking, memory protection, full symmetric multiprocessing, and high-performance I/O.

Rhapsody's core operating system will eventually support a variety of file and volume formats, including HFS, UFS, DOS FAT, ISO9660, AFP, and HFS+ (an enhanced version of HFS).

Rhapsody: Where OpenStep Fits In

Rhapsody is a ensemble of technologies in which OpenStep plays a central role. The diagram below depicts Rhapsody on PowerPC-based systems; it shows an enhanced and expanded version of OpenStep as the *Yellow Box*. The Yellow Box has interfaces to the core operating system, to the advanced Macintosh user experience, to a Java virtual machine, and to a Mac OS–compatibility subsystem known as the *Blue Box*.

Blue Box. This is a native Mac OS environment that runs on the Rhapsody kernel as another application (on PowerPC–based systems only). It enables the execution of programs written for current and prior versions of the Mac OS. Because a Blue Box environment is "just" another Rhapsody application, it shares in the benefits of the kernel. For example, one or more Blue Boxes can be running at a time on one machine; if one crashes, it will not affect other Blue Boxes or any other Rhapsody application. A Blue Box can take over the entire screen or it can occupy only a portion of it. In full-screen mode you can use a hot key to switch between the Blue and Yellow Boxes.

Mac OS software running in the Blue Box cannot directly access services provided by the core operating system or the Yellow Box; by the same token, programs written for the Yellow Box cannot directly access services in the Blue Box. Rhapsody supports limited sharing of data between Blue and Yellow Boxes, including copy-and-paste, and will eventually permit communication between them through Apple events.

Some Yellow Box frameworks and development tools are being ported to the Blue Box, allowing Yellow Box development in that environment.

Many current Mac OS technologies, such as QuickDraw 3D, QuickDraw GX, OpenDoc, and QuickTime, are being carried forward into the Blue Box.

Advanced look and feel. The new Rhapsody operating system sports a new exterior that perfects the Mac OS, itself legendary for ease of use. The Rhapsody user experience combines the best visual elements and usage models of the Mac OS and of OPENSTEP for Mach, and it incorporates new paradigms in human-interface design. Yet it is still recognizably a Macintosh operating system.

Programs written for the Yellow Box and Java programs written to the Yellow Box APIs will present the same advanced Macintosh look and feel to users.

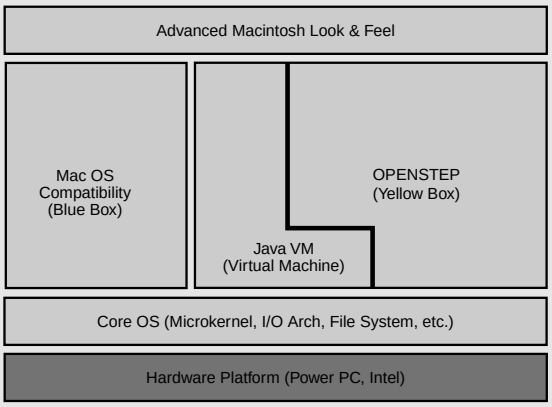
Java virtual machine. Recognizing the increasing importance of Java in software technology, Apple is including a Java virtual machine (VM) as part of the Rhapsody picture. The Yellow Box will offer APIs in Java for accessing the functionality of the core frameworks, and it will include ported versions of the AWT, IFC, AFC, and other Java packages. With these possibilities, you can create Java programs that will run on any platform with a Yellow Box or "100% pure" Java applications that will run on any platform that has a Java virtual machine.

Rhapsodic Variations

With little more than a recompile, you can deploy applications written to the Yellow Box APIs (aka OpenStep) in four different user environments:

- **Rhapsody** (for PowerPC)**.** See diagram below.
- **Rhapsody for Intel Processors**. Same as above, minus the Blue Box.
- **Yellow Box for Windows**. For Windows NT and Windows 95.
- **Yellow Box for Mac OS**. (Forthcoming.)

Applications will exhibit the "look and feel" appropriate to the underlying operating system.



The Yellow Box: A Blend of Technologies

What makes Rhapsody a uniquely rich environment for both users and developers is the assortment of technologies clustered around the core OpenStep frameworks; taken together, they are known as the Yellow Box. Some of these extended frameworks are NeXT products, carried forward; some frameworks are being developed for Rhapsody; and others are being ported from the current Mac OS—QuickTime, QuickDraw 3D, QuickDraw GX, and ColorSync, to name a few.

The sections that follow describe the range of technologies that these extended frameworks will make available. They also summarize some the fundamental technologies incorporated by Rhapsody.

Imaging and Printing Model

The imaging *and* printing model for Rhapsody is Display PostScript. Unlike in the Mac OS, the same mechanism is used to view and print what appears on the screen. You no longer have to duplicate code to send output both to the screen and to PostScript-based devices. The best of Apple's graphic technologies, including ColorSync and QuickDraw GX typography, is being migrated to the Display PostScript model.

Display PostScript has several other advantages over existing Mac OS imaging models. It improves the performance and responsiveness of the graphical user interface. It is a widely accepted standard in the industry. It is also easy to write code for; you use the Graphics APIs or the APIs of the Display PostScript framework. However, applications that must severely limit the overhead for imaging can access the frame buffer directly by using APIs for this purpose, called Interceptor.

Multimedia

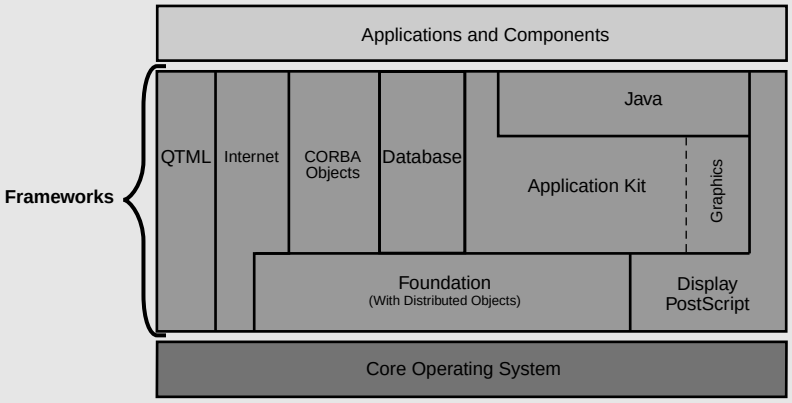
The QuickTime Media Layer (QTML) is a collection of objects and APIs that make it easy for you to give applications rich multimedia content. QTML consists of three separate products:

- QuickTime is the industry-standard multimedia architecture that software vendors and content creators use to store, edit, and play synchronized graphics, sound, video, text, and music.
- QuickTime VR delivers virtual reality in both panoramas and objects.
- QuickDraw 3D enables developers to render real-time three-dimensional graphics.

Distributed Computing

The Yellow Box offers APIs for creating distributed applications, thereby eliminating the need to write code for many low-level network operations. The Foundation framework includes Distributed Objects, a technology that permits objects in different task or threads—on a single host or across a network—to communicate with each other.

The Yellow Box will also support CORBA/IIOP (including an object request broker), making it easy to create industrial-strength applications that work across different types of networks.



Internet

Apple intends Rhapsody to be a major technological force in the world of the Internet. The development platform features APIs for Internet-based mail, messaging, directories, and security services. Moreover, the Yellow Box includes the WebObjects framework which, along with WebObjects Builder (an interactive application for the creation of dynamic Web pages), enables the speedy prototyping and development of dynamic Web-server applications. These applications can access data in standard relational databases and can communicate with applet components on the client browser. The Yellow Box also offers built-in HTML rendering capabilities.

Database Integration

The Enterprise Objects Framework provides applications with access to data on local relational database systems. Adaptors for Oracle, Sybase, and Informix, and ODBC-compliant databases are available separately. Enterprise Objects achieves persistent storage of data through a process of mapping objects to tables.

The Yellow Box will also make a local database engine available, allowing you to build and test database applications locally. When these applications are deployed in a client/server environment, the customer then needs to acquire a Web-application server (WebObjects) or to get an adaptor for the supported databases.

Localization and Internationalization

You can easily localize Yellow Box applications largely because of a well-designed localization architecture and Unicode support, both built into the Application Kit. In this architecture, user-interface elements—as archived objects and as resources—are kept separate from the executable. It's therefore possible to have a single code base that is qualified for various locales. You can even have multiple localizations bundled with one application, greatly reducing the overall footprint of an application in its various localizations. Since localization bundles can be easily added to or removed from an existing application, new localizations can be distributed through updaters.

Because the Yellow Box uses Unicode 2.0 as its native character set, applications can easily handle all of the world's languages. The prevalence of Unicode also eliminates many character-encoding hassles. Still there are Yellow Box APIs to help translate between Unicode and other major character sets in use today.

Apple's development environment supports localization in several important ways. It gives you an easy way to identify which files are to be localized (and for which language). Andit enables you to create a series of archivable user interfaces, each designed for a particular locale.

Text and Fonts

The Yellow Box offers a powerful set of text services that can be readily adapted by text-intensive applications requiring high performance. These services, which can support text buffers larger than 32K, include kerning, ligatures, tab formatting, and rulers.

By the Unified Release, the Yellow Box will support a variety of font formats, including Type 1 PostScript, Type 3 PostScript, Type 42 PostScript, and TrueType (including the typographic capabilities of TrueType GX). The goal for Rhapsody is for an open font architecture that makes it easy for users to work with any font format they want.

Microsoft Windows

You can develop a Yellow Box application on one platform, say Rhapsody for Power PC, and deploy it on another supported platform, including Windows, with little more than a recompilation. Yellow Box applications that run on Microsoft Windows have a range of capabilities at their disposal. With OLE/COM, Yellow Box applications can transparently communicate with OLE-enabled applications such as Microsoft Word. They can also use ActiveX controls within the Windows environment. OLE/COM and ActiveX support—as well as the ability to make Win32 calls from your code—permits the development of industrial-strength Windows applications that are seamlessly integrated with other Windows applications.

Component Technologies

One of the key advantages of the Yellow Box as a development environment is the capability for developing programs quickly by assembling reusable components. With the proper programming tools and a little work, you can build Yellow Box components that can be packaged and distributed for others to use. Applications are an obvious example of this component technology, but there are others. With the Yellow Box and Apple's development tools, you can create:

- Frameworks, which other developers can use to create programs by writing code based on the framework APIs
- Bundles containing executable code and associated resources, which programs can dynamically load
- Palettes containing custom user-interface objects that other developers can drag and drop into their own user interfaces using the Rhapsody development tools

With the Yellow Box's component architecture, you can easily create and distribute extensions and plug-ins for applications. You can also develop components based on JavaBeans technology in both the Yellow Box and the Blue Box. JavaBeans components will be integrated into the Rhapsody development tools. On Windows you can use ActiveX components.

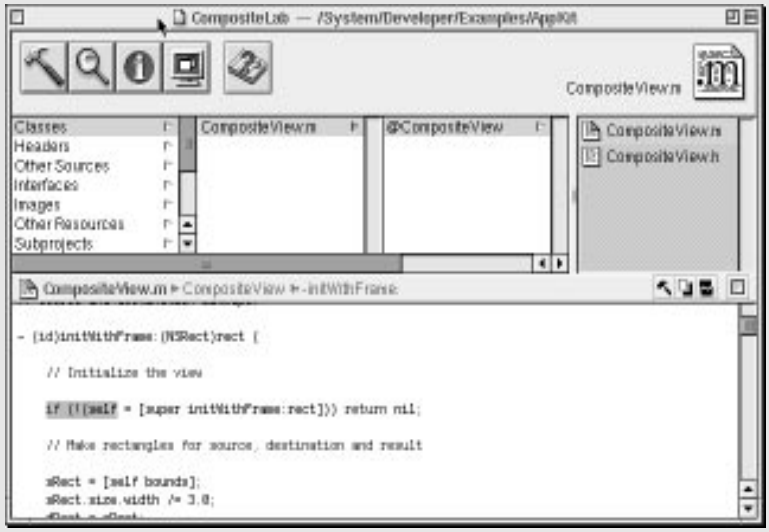
Programming in the Apple Development Environment

Apple has a powerful, integrated, cross-platform development environment for the Yellow Box on Rhapsody. With it you can easily build applications, and easily deploy them on multiple platforms: Windows NT, Windows 95, and (of course) Rhapsody on both PowerPC–based and Pentium-based machines. You develop the application for one platform and, with little more than a simple recompile, the same application is ready to run on another platform. You can also develop Rhapsody applications written in Java which will run on any platform supporting Apple's Java virtual machine (see "Java Looming" on the following page).

The Apple development environment consists of a suite of applications and tools that deliver maximum productivity from the frameworks, subsystems, libraries, components, and other resources of Rhapsody. The principal applications are Interface Builder and Project Builder.

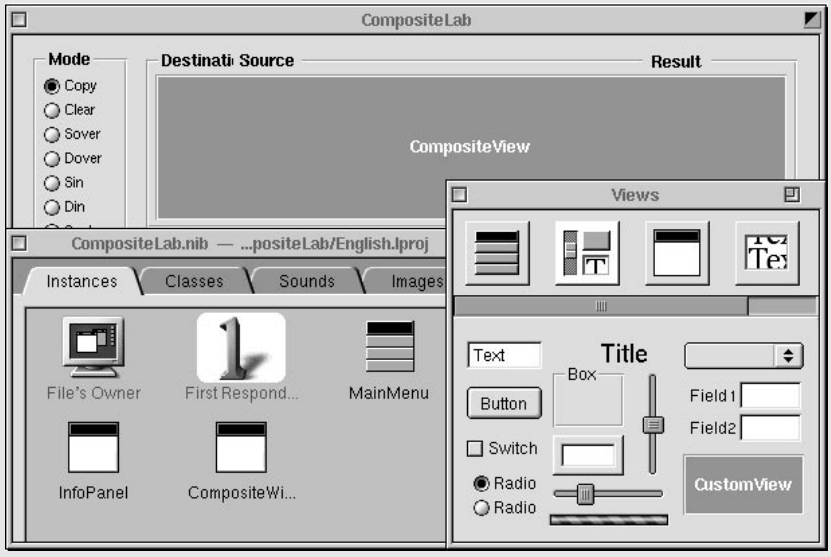
Project Builder is an application that manages software-development projects and that orchestrates and streamlines the development process. It integrates a project browser, a full-featured code editor, language-savvy symbol recognition, a class browser, sophisticated project search capabilities, header file and documentation access, build customization, a graphical debugger, Java support, and a host of other features.

Project Builder allows you to plug in compilers from other tool vendors (such as Metrowerks). It facilitates the creation of different types of projects (such as palettes, frameworks, and bundles, in addition to applications). It also provides programmatic hooks for integrating source-control management systems.



Interface Builder makes it easy to create application interfaces. You just drag an object from a palette and drop it on the graphical user interface you're creating. You can then set attributes of these objects through an inspector panel and you can connect them to other objects in your application so they can send messages to each other. Interface Builder also assists in the definition of custom classes and allows you to test an interface without having to compile a line of code.

Standard palettes hold an assortment of Application Kit objects. Other palettes can include Yellow Box objects from other frameworks, third-party objects, and custom compiled objects. You can also store non-compiled configurations of objects on *dynamic palettes*. Interface Builder archives and restores elements of a user interface (including connections) as objects—it doesn't "hardwire" them into the interface. Interface Builder also enables youto connect your application to JavaBeans and ActiveX components.



Other Development Tools

Apple's development environment for the Yellow Box has much more than Project Builder and Interface Builder to offer. There are other applications: FileMerge, which allows you to compare and selectively merge files and directories; Yap, which allows you to preview and test PostScript code; and MallocDebug and other applications that analyze and optimize code that you've written.

In addition, the development environment includes a shell application (**Terminal.app**) with which you can run many development utilities. However, use of command-line utilities is optional; they are not required for Yellow Box development. (Eventually, many of these utilities will be incorporated into new development applications.)

Pick Your Language

In developing applications for Rhapsody's Yellow Box, you have a choice of programming language. You can write programs, in whole or in part, in C, C++, Java, and Objective-C. Soon scripting languages will also be supported.

Some developers have the notion that Objective-C is difficult. They are mistaken. You shouldn't dread the thought of learning Objective-C. It is a simple and elegant language. A typical developer, especially one experienced in C++, should need no more than a day or two to learn Objective-C.

Note: Yellow Box programs cannot be completely written in C++ because you cannot subclass Yellow Box classes in C++. For your program to take advantage of the Yellow Box frameworks, C++ objects must be integrated with Objective-C or Java objects.

Java Looming

Apple is aware of the growing importance of Java and expects that Java will become the language of choice for many developers. Since Java is central to Apple's system and development strategies, both the Blue and Yellow Boxes will feature high-performance Java virtual machines and will include the latest versions of the Java Development Kit (JDK).

Besides hosting native Java packages—including AWT, JFC, AFC, and IFC—the Yellow Box will provide access to its own APIs in Java. You will be able to subclass Yellow Box classes in Java and mix "pure" Java and Yellow Box objects in your code. You will be able to write "100% pure" Java applications that can run on any platform that has the appropriate virtual machine. Or you can write Java applications that use Yellow Box APIs; these applications can run—without recompiling—anywhere the Yellow Box is available. Rhapsody will also integrate JavaBeans into the Yellow Box run time and into Interface Builder palettes.

Project Builder will include several features that specifically support the development of Java applications.

How Apple's Development Environment Compares

The Yellow Box development environment is, in several ways, like traditional Macintosh development environments such as MPW and CodeWarrior. For example, like these products, it permits a great deal of customization, and it possesses sophisticated searching capabilities. But the Yellow Box development environment is different in many ways, both large and small. Some of the differences take some getting used to, such as the bindings of shortcut keys and the way you browse the project.

Apple is listening to its developers and is rapidly evolving its development tools to meet the needs and expectations of this community. In addition, traditional vendors of Macintosh development tools, such as MetroWerks, are busy porting existing tools to Rhapsody or are creating new tools. For example, MetroWerks is developing a tool (Latitude) that will assist Macintosh developers in porting their PowerPlant and other projects to the Yellow Box. Eventually, Rhapsody developers will be able to "mix and match" the exact tool set that suits them best.

Programming With Objects

For some Mac OS developers, the most striking disparity they'll experience when they start developing Yellow Box programs is not the tool set. It is the shift in mind set that is required for object-oriented programming (OOP).

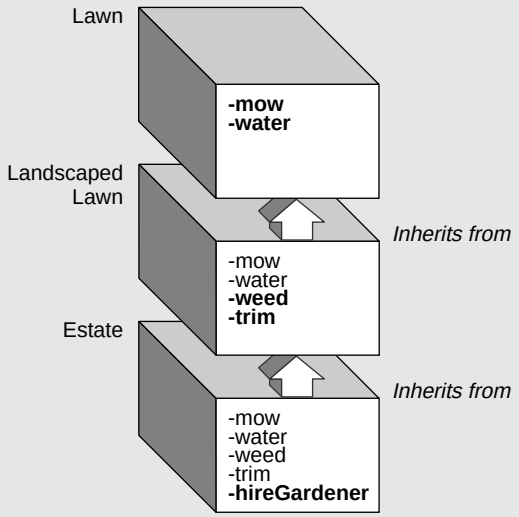
Instead of thinking in terms of procedures and data, you have to think in terms of objects—discrete programmatic units containing their own data as well as procedures that act on that data. An object-oriented program is composed of objects of different types, each type fulfilling a specialized role within the program. In such a program, objects are constantly sending messages to each other—that is, they are requesting other objects to execute a procedure. The object receiving the message performs what is requested and, in some cases, returns an object or another value. (For more on how object-oriented programs works, see the appendix "Object-Oriented Programming.")

Learning how to program with objects takes some initial effort, but with some familiarity, object-oriented programming begins to seem natural, elegant, and powerful. And, with the rich functionality of the Yellow Box frameworks to tap, application development becomes easier—you get many application features "for free." Programming with objects, especially Yellow Box objects, increases your productivity by freeing you from many repetitive coding tasks. You have more time to accomplish what is truly creative.

To mesh your custom code with framework objects, you must create a subclass of at least one of the framework classes. The subclass implements behavior or logic specific to your application and obtains the services it needs from framework objects. Moreover, a custom subclass inherits attributes and behavior from its superclass, again without you having to write a single line of code (see illustration). Often one or two subclasses is all that is required to achieve quite substantial results.

Of course, to program effectively with the Yellow Box you must learn what services you can obtain from framework classes and what attributes and behavior you inherit from them. Even for developers experienced in object-oriented programming, the Yellow Box frameworks pose their own learning hurdle. You need to become familiar with the class hierarchy, to discover what classes can do, and learn how they interact with one another.

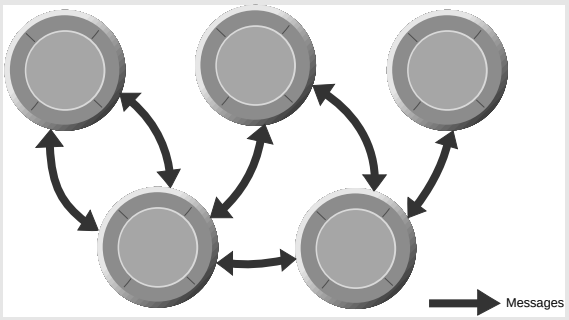
This learning requirement is unavoidable, regardless of the development environment. But Apple tries to ease the effort required with tool features such as a class browser and documentation such as the framework references and this book, which introduces some of the more fundamental classes.



The Advantage of Objects

Object-orientation is the software equivalent of the Industrial Revolution. In the same way that modern factories assemble products out of prefabricated components rather than manufacture every product from scratch, object-orientation allows programmers to build complex software by reusing software components called objects. Specifically, objects lead to several measurable advantages:

Greater reliability. By breaking complex software projects into small, self-contained, and modular objects, object-orientation ensures that changes to one part of a software project will not adversely affect other portions of the software. Being small, each of these objects is a well-tested module of code, and so the overall reliability of the software increases.

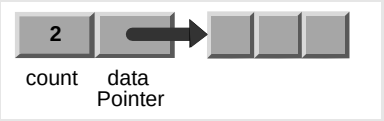


Easier maintainability. Since objects are modular and usually small (in terms of the overall code size of a project), bugs in code are easier to locate. Developers can also change the implementation of an object without causing havoc in other parts of an application.

Greater productivity through reuse. One of the principal benefits of object-orientation is reuse. One object can be integrated into many applications. And through subclassing, you can create specialized objects merely by adding the code unique to the new object. Objects of the new subclass inherit functionality from the superclass, reducing coding and promoting greater reliability.

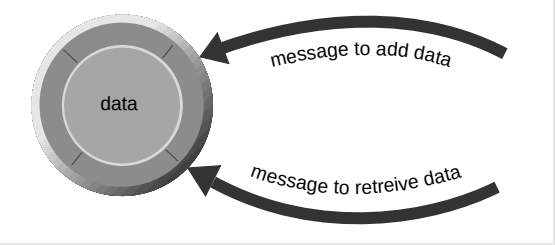
An Example

Object-oriented programming delivers its greatest benefits to large and complex programs. But its advantages can also be demonstrated with a simple data structure such as might be used in any application.



With procedural programming techniques, the application is directly responsible for data manipulation. One problem with this is illustrated in the picture above. It shows a data structure consisting of a **count** variable and a data pointer. Since the application directly manipulates the data, it has the opportunity to introduce inconsistencies. Here, it has added an item to the data, but has forgotten to increment the count; the **count** variable says there are still only two data elements when in fact there are three. The structure has become inconsistent and unreliable.

Another problem is that all parts of the application must have intimate knowledge about the structure of the data. If the allocation of data elements is changed from a statically allocated array to a dynamically allocated linked list, it would affect every part of the application that accesses, adds, or deletes elements from the list.



With an object-oriented programming paradigm, the application as a whole doesn't directly manipulate the data structure; rather, that task is entrusted to a particular object. Since the application doesn't directly access the data, it can't introduce inconsistencies. Note also that it's possible to change the implementation of the object without breaking other parts of the application. For example, the data storage method could be changed to optimize performance. So long as the object responds to the same messages, other parts of the application are unaffected by internal implementation details.