This chapter describes the interapplication communication (IAC) architecture for Macintosh computers, summarizes how your application can take advantage of it, and tells you where in this book to find the information you need to perform specific tasks.

The Apple Event Manager, Event Manager, and Program-to-Program Communications (PPC) Toolbox underlie all the IAC tasks your application can perform. This chapter introduces the Apple Event Manager and the Program-to-Program Communications Toolbox. For information about the Event Manager, see *Inside Macintosh: Macintosh Toolbox Essentials.* For definitions of the standard Apple events available for use by all applications, see the *Apple Event Registry: Standard Suites.*

The IAC architecture includes the Open Scripting Architecture (OSA). The OSA provides a mechanism that allows users to control multiple applications by means of scripts, or sets of instructions, written in a variety of scripting languages. Each scripting language has a corresponding scripting component that is managed by the Component Manager. When a user executes a script, the scripting component sends Apple events to one or more applications to perform the actions the script describes.

This chapter introduces the OSA and describes how to make your application scriptable, or capable of responding to Apple events sent to it by a scripting component. For more information about using the Component Manager, see *Inside Macintosh: More Macintosh Toolbox.*

Overview of Interapplication Communication

The ***interapplication communication (IAC) architecture*** provides a standard and extensible mechanism for communication among Macintosh applications. The IAC architecture makes it possible for your application to

- provide automated copy and paste operations between your application and other applications
- be manipulated by means of scripts
- send and respond to Apple events
- send and respond to high-level events other than Apple events
- read and write blocks of data between applications

The chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* describes how your application can use Event Manager routines to send and respond to high-level events. High-level events need not adhere to any specific protocol, so their interpretation is defined by each application that sends or receives them.

The most important requirement for high-level communication among all applications is a common vocabulary of events. To provide such a standard, Apple Computer, Inc., has defined a protocol called the ***Apple Event Interprocess Messaging Protocol (AEIMP).*** High-level events that conform to this protocol are called ***Apple events.***

The vocabulary of publicly available Apple events is published in the *Apple Event Registry: Standard Suites*, which defines the standard Apple events that developers and Apple have worked out for use by all applications. To ensure that your application can communicate at a high level with other applications that support Apple events now and in the future, you should support the standard Apple events that are appropriate for your application.

Effective IAC requires close cooperation among applications at several levels. In addition to the format for high-level events and the standard vocabulary of Apple events, Apple has defined several other standards your application can use to communicate with other applications. These include standard methods for dealing with shared dynamic data, scripts, and low-level message blocks.

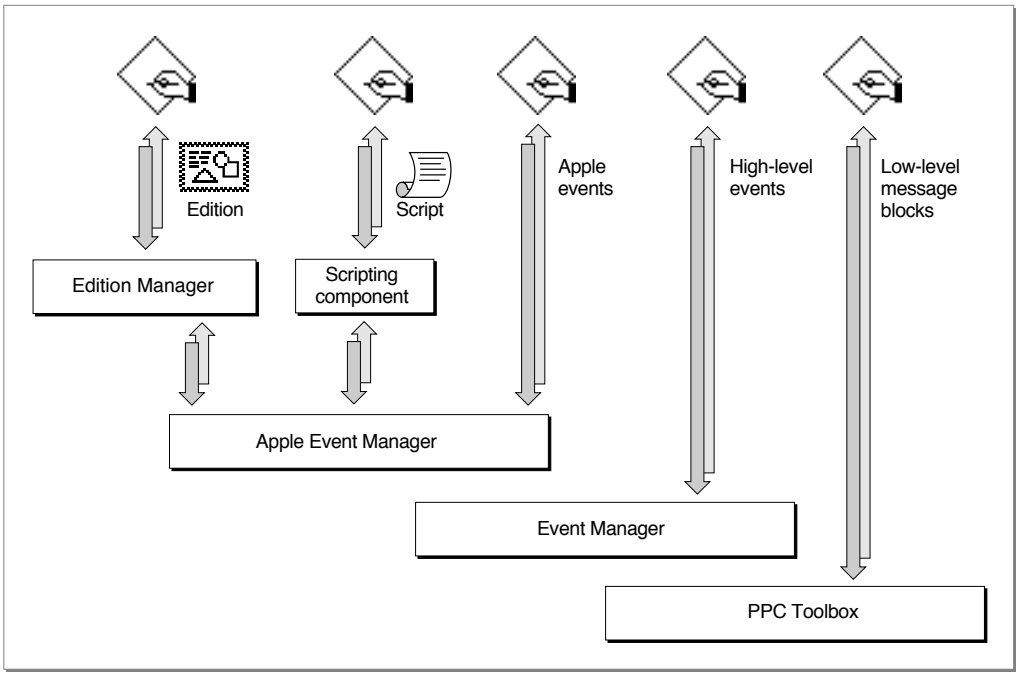
The IAC architecture comprises the following parts:

- The ***Edition Manager*** allows applications to automate copy and paste operations between applications, so that data can be shared dynamically.
- The ***Open Scripting Architecture (OSA)*** provides a standard mechanism, based on the Apple Event Manager, that allows users to control multiple applications by means of scripts written in a variety of scripting languages.
- The ***Apple Event Manager*** allows applications to send and respond to Apple events.
- The ***Event Manager*** allows applications to send and respond to high-level events other than Apple events.
- The ***Program-to-Program Communications (PPC) Toolbox*** allows applications to exchange blocks of data with each other by reading and writing low-level message blocks. It also provides a standard user interface that allows a user working in one application to select another application with which to exchange data.

Figure 1-1 shows the primary relationships among these parts. The managers and components toward the top of the figure rely on the managers beneath them. The Edition Manager uses the services of the Apple Event Manager to support dynamic data sharing. Scripting components manipulate and execute scripts with the aid of the Apple Event Manager. The Apple Event Manager in turn relies on the Event Manager to send Apple events as high-level events, and the Event Manager uses the services of the PPC Toolbox.

Figure 1-1 also shows the five principal means of communication provided by the IAC architecture. In addition to using the Edition Manager and scripting components to send Apple events on their behalf, applications can use the Apple Event Manager directly to send Apple events to other applications. All applications can use the Apple Event Manager to respond appropriately to Apple events, whether they are sent by the Edition Manager, a scripting component, or other applications. Applications can also use the Event Manager directly to send or receive high-level events other than Apple events, and the PPC Toolbox directly to send or receive blocks of data.

Figure 1-1 Principal methods of communication between applications



The five forms of IAC shown in Figure 1-1 can be summarized as follows:

- *Sharing dynamic data.* The Edition Manager allows users to copy data from one application's document to another application's document, updating information automatically when the data in the original document changes. The verbs *publish* and *subscribe* describe this form of dynamic data sharing, and the noun *edition* describes a copy of the data to be shared. Applications that support dynamic data sharing must implement the Create Publisher and Subscribe To menu commands. The Edition Manager provides the interface that allows applications to share editions.

  You can let users publish and subscribe on a local volume or across a network. In general, users should be able to publish or subscribe to anything that they can copy or paste. "Sharing Data Among Applications," which begins on page 1-6, describes how you can use the publish and subscribe features in your application.

- *Scripting.* The OSA includes the Apple Event Manager, the Apple events defined by the *Apple Event Registry: Standard Suites,* and the routines supported by ***scripting components,*** which applications can use via the Component Manager to execute scripts. Script-editing applications such as Script Editor (not shown in Figure 1-1) allow users to manipulate and execute scripts.

  Each scripting language has a corresponding scripting component that can execute scripts written in that language. Scripting components typically implement a text-based scripting language based on Apple events. For example, the ***AppleScript component*** implements ***AppleScript,*** the standard user scripting language defined by

Apple Computer, Inc. When the AppleScript component executes a script, it performs the actions described in the script, including sending Apple events to applications when necessary.

"Supporting AppleScript and Other Scripting Languages," which begins on page 1-13, describes how the OSA makes it possible for your application to

- provide human-language equivalents to Apple event codes so that scripting components can send your application the appropriate Apple events during script execution
- allow users to record their actions in the form of a script
- manipulate and execute scripts

- *Sending and responding to Apple events.* Your application can send Apple events directly to other applications to request services or information or to provide information. To support AppleScript and most other scripting languages based on the OSA, your application must be able to respond to Apple events. "Sending and Responding to Apple Events," which begins on page 1-9, describes how applications can send and respond to Apple events with the aid of the Apple Event Manager.
- *Sending and responding to other high-level events.* The Event Manager allows applications to support high-level events other than Apple events. See the chapter "Event Manager" in *Inside Macintosh: Macintosh Toolbox Essentials* for information about high-level events.
- *Exchanging message blocks.* The PPC Toolbox allows applications to exchange blocks of data with each other by reading and writing low-level message blocks. This method of communication is most useful for applications that are closely integrated, specifically designed to work together, or dependent on each other for information. It can also be used in code that is not event-based. See "Exchanging Message Blocks" on page 1-22 for a summary of the capabilities provided by the PPC Toolbox.

All forms of IAC are based on the premise that applications cooperate with each other. Both the application sending a high-level event or low-level message block and the application receiving it must agree on the protocol for communication. You can ensure effective high-level communication between your application and other Macintosh applications by supporting the standard Apple events defined in the *Apple Event Registry: Standard Suites.*

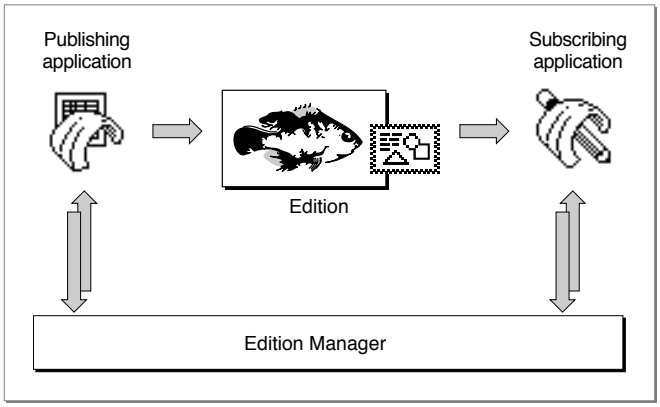
Sharing Data Among Applications

All Macintosh applications can use the Scrap Manager to share static data by allowing the user to copy and paste data between documents. Dynamic data sharing, or automated copy and paste operations between applications, extends this capability to dynamically changing data. The Edition Manager lets applications share dynamic data at the user's request. You incorporate publish and subscribe capabilities in your application much as you incorporate copy and paste capabilities.

A user can publish data by selecting a portion of text, graphics, or other data in a document and choosing Create Publisher from the Edit menu. In response, your application saves the selected information in a separate file. This stored information is referred to as an ***edition.*** The user can subscribe to an edition by choosing Subscribe To from the Edit menu; when the user selects a file that contains an edition, your application includes the information from the edition in the current document. The information in an edition can be shared by many documents.

Figure 1-2 shows the principal relationships among the Edition Manager, the publishing application, the subscribing application, and the file that contains the edition. In addition to the relationships illustrated in the figure, the Edition Manager uses the Apple Event Manager to communicate with applications that are sharing dynamic data.

Figure 1-2 Sharing data with the aid of the Edition Manager

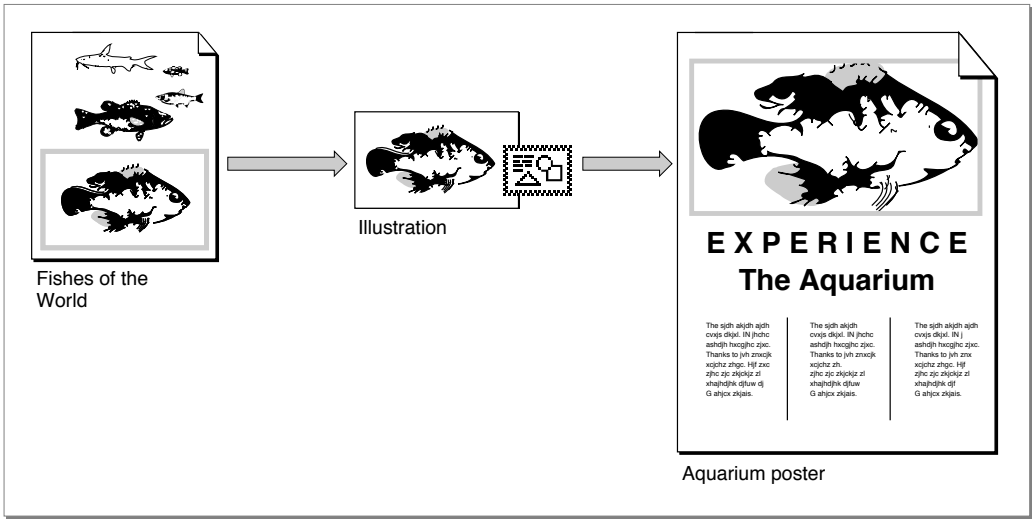


A ***publisher*** is a portion of a document that is made available to other documents through an edition. A ***subscriber*** is a portion of a document that reads the information from an edition.

Figure 1-3 shows a document containing a publisher, a file containing an edition, and a document containing a subscriber. The bottom fish in the Fishes of the World document is a publisher. The information from this publisher is made available to other documents through the Illustration edition. The Aquarium poster document contains a subscriber that gets its information from the Illustration edition. Note that when a user selects a publisher or subscriber within a document, your application should display a border surrounding the publisher or subscriber.

In general, when a user modifies the contents of a publisher and saves the document, your application should write the new data to the edition. The Edition Manager then uses the Apple Event Manager to inform all open applications with subscribers to the edition that it has been updated. These applications can then automatically update the subscribers in the documents.

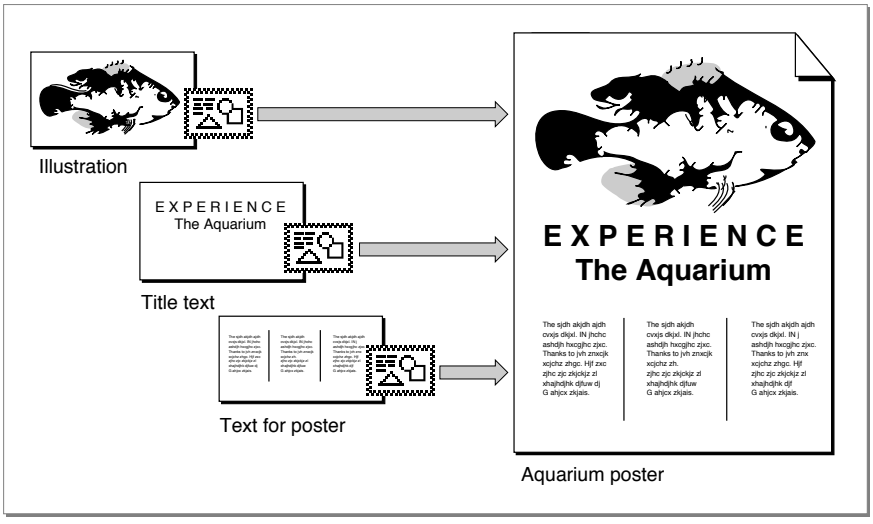
Figure 1-3 A publisher, an edition, and a subscriber



For example, suppose the user changes the color of the fish in the Fishes of the World document shown in Figure 1-3, then saves the document. This automatically changes the Illustration edition, and the subscribing application can update the Aquarium poster document if that's what the user wants to do.

Figure 1-4 shows how a user might create a poster from information contained in other documents.

Figure 1-4 Sharing dynamic data with other applications



Your application should save the new information in the edition whenever the user edits the publisher and saves the document that contains the publisher—unless the user has indicated that the information should be saved in the edition on request only. When the user saves new information in an edition, the Edition Manager replaces the previous contents.

When an edition is updated, the Edition Manager informs your application. Your application should then update any subscribers (unless the user has indicated that updates should be incorporated on request only).

For example, suppose a user opens a word-processing document called My Stocks that accesses information from an edition called Stock Report. The Stock Report edition might be updated twice a day by an online database. As the information in the edition changes, the My Stocks document can receive automatic updates with the latest information.

You can implement publish and subscribe capabilities in your application by using the routines provided by the Edition Manager and supporting the related Apple events. The chapter "Edition Manager" in this book provides sample code that shows how to add these features to your application. The chapter "Responding to Apple Events" in this book describes how to support the related Apple events.

Sending and Responding to Apple Events

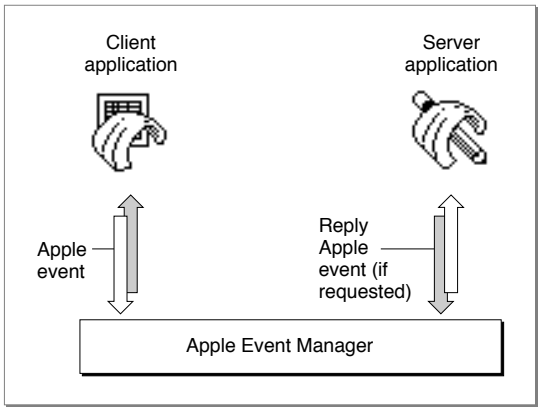
An Apple event is a high-level event that conforms to the Apple Event Interprocess Messaging Protocol. The Apple Event Manager uses the Event Manager to send Apple events between applications on the same computer or between applications on remote computers.

Applications typically use Apple events to request services and information from other applications or to provide services and information in response to such requests. For example, any application can use the Get Data Apple event to request that your application locate and return a particular set of data, such as a table. If your application supports the Get Data event, it should be able to recognize the event and respond by locating the requested data and returning a copy of the data to the application that requested it.

Communication between two applications that support Apple events is initiated by a ***client application,*** which sends an Apple event to request a service or information. For example, a client application might request services such as printing specific files, checking the spelling of a list of words, or performing a numeric calculation; or it might request information, such as one customer's address or a list of names and addresses of all customers living in Ohio. The application providing the service or the requested information is called a ***server application.*** The client and server applications can reside on the same local computer or on remote computers connected to a network.

Figure 1-5 shows the relationships among a client application, the Apple Event Manager, and a server application. The client application uses Apple Event Manager routines to create and send the Apple event, and the server application uses Apple Event Manager routines to interpret the Apple event and respond appropriately. If the client application so requests, the server application adds information to a reply Apple event that the Apple Event Manager returns to the client application.

Figure 1-5 Sending and responding to Apple events with the aid of the Apple Event Manager



If an Apple event is one of the standard events defined in the *Apple Event Registry: Standard Suites*, the client application can construct the event and the server application can interpret it according to the standard definition for that event. To ensure that your application can respond to Apple events sent by other applications, you should support the standard Apple events that are appropriate for your application.

Standard Apple Events

The current edition of *Apple Event Registry: Standard Suites* defines the standard ***suites*** of Apple events, which are groups of related events that are usually implemented together. The Apple Event Registrar maintains the *Apple Event Registry: Standard Suites* and other information about the ongoing development of Apple event suites.

The standard suites include the following:

- *The Required suite* consists of the four Apple events that the Finder sends to applications. These events are Open Application, Open Documents, Print Documents, and Quit Application. The Finder uses the required events as part of the mechanisms in System 7 and later versions for launching and terminating applications. To support System 7, your application must support the required Apple events as described in the chapter "Responding to Apple Events" in this book.

- *The Core suite* consists of the basic Apple events, including Get Data, Set Data, Move, Delete, and Save, that nearly all applications use to communicate. You should support the Apple events in the Core suite that make sense for your application.
- *A functional-area suite* consists of a group of Apple events that support a related functional area. Functional-area suites include the Text suite and the Database suite. You can decide which functional-area suites to support according to which features your application provides. For example, most word-processing applications should support the Text suite, and most database applications should support the Database suite.

You do not need to implement all Apple events at once. You should begin by supporting the required Apple events, then add support for the events sent by the Edition Manager, the core events, and the functional-area events as appropriate for your application.

If necessary, you can extend the definitions of the standard Apple events to suit specific capabilities of your application. You can also define your own custom Apple events. However, only those applications that choose to support your custom Apple events explicitly will be able to make use of them. If all applications communicated solely by means of custom Apple events, every application would have to support all other applications' custom events. Instead of creating custom Apple events, try to use the standard Apple events and extend their definitions as necessary.

Apple events describe actions to be performed by the applications that receive them. In addition to a vocabulary of actions, or "verbs," effective communication between applications requires a method of referring to windows, data (such as words or graphic elements), files, folders, volumes, zones, and other items on which actions can be performed. The Apple Event Manager provides a method for specifying structured names, or "noun phrases," that applications can use to describe the objects on which Apple events act.

The *Apple Event Registry: Standard Suites* includes definitions for ***Apple event object classes,*** which are simply names for objects that can be acted upon by each kind of Apple event. Applications use these definitions and Apple Event Manager routines to create complex descriptions of almost any discrete item in another application or its documents. For example, an application could use Apple Event Manager routines and standard object class definitions to construct a Get Data event that requests "the most recent invoice to John Chapman in the Invoices database on the Archives server in the Accounting zone" and send the event to the appropriate application across the network.

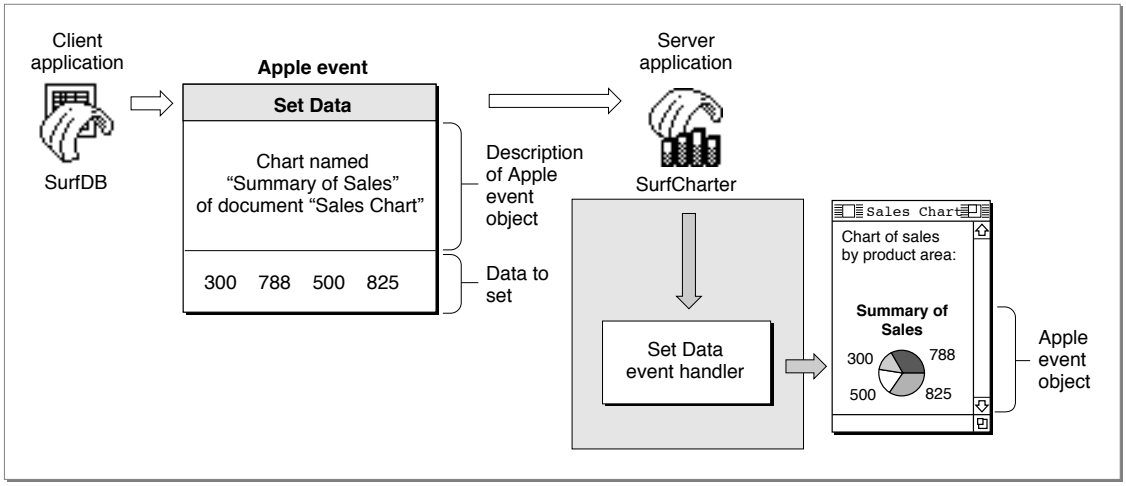
An ***Apple event object*** is any item supported by an application, such as a word, paragraph, shape, or document, that can be described in an Apple event. In the example just given, the specified invoice, the Invoices database, the Archives server, and the Accounting zone are nested Apple event objects. Nearly any item that a user can differentiate and manipulate on a Macintosh computer can be described as an Apple event object of a specified object class nested within other Apple event objects. When handling an Apple event that includes such a description, an application must locate the specified Apple event object and perform the requested action on it.

Most of the standard Apple events defined in the *Apple Event Registry: Standard Suites* require your application to recognize specific Apple event object classes. Support for the standard Apple events, including Apple event object classes, allows your application to respond to requests for services or information from any other application or process.

Handling Apple Events

Figure 1-6 shows a common Apple event from the Core suite, the Set Data event. The SurfDB application is the client; it sends a Set Data event to the SurfCharter application. This event requests that SurfCharter use some new sales figures generated by SurfDB to update the data for the chart named "Summary of Sales" in the document named "Sales Chart." The Apple event contains information that identifies an action—setting data—and a description of the Apple event object on which to perform the action—"the chart named Summary of Sales in the document named Sales Report." The Apple event also includes the new data for the chart.

Figure 1-6 A Set Data event



To respond appropriately, the SurfCharter application in Figure 1-6 can use the Apple Event Manager to determine what kind of Apple event has been sent and pass the event to the appropriate Apple event handler. An ***Apple event handler*** is an application-defined function that extracts pertinent data from an Apple event, performs the requested action, and returns a result. In this case, the Set Data event handler must locate an Apple event object—that is, the specified chart in the specified document—and change the data displayed in the chart as requested.

The Apple Event Manager provides routines that a server application can use in its Apple event handlers to take apart an Apple event and examine its contents. The SurfCharter application in Figure 1-6 can interpret the contents of the Set Data Apple event according to the definition of that event in the *Apple Event Registry: Standard Suites.* The Set Data event handler uses both Apple Event Manager routines and the SurfCharter application's own routines to locate the chart and make the requested change.

The Apple Event Manager also provides routines that a client application can use to construct and send an Apple event. However, the most important requirement for applications that support IAC is the ability to respond to Apple events, because this ability is essential for an application that users can control through scripts. The next section describes how you can use Apple events to support scripting in your application.

The chapter "Introduction to Apple Events" in this book provides an overview of Apple events and describes how you can use the Apple Event Manager to implement Apple events in your application. The chapters "Responding to Apple Events," "Creating and Sending Apple Events," "Resolving and Creating Object Specifier Records," and "Recording Apple Events" provide detailed information about the Apple Event Manager.

Supporting AppleScript and Other Scripting Languages

A ***script*** is any collection of data that, when executed by the appropriate program, causes a corresponding action or series of actions. For example, some database, telecommunications, and page-layout applications allow users to automate repetitive or conditional tasks by means of scripts written in proprietary scripting languages. The HyperTalk® scripting language allows users to control the behavior of HyperCard® stacks. Macro programs can automate tasks at the level of mouse clicks and keystrokes.

The Open Scripting Architecture (OSA) provides a standard mechanism that allows users to control multiple applications with scripts written in a variety of scripting languages. Each scripting language has a corresponding scripting component. When a scripting component executes a script, it performs the actions described in the script, including sending Apple events to applications if necessary.

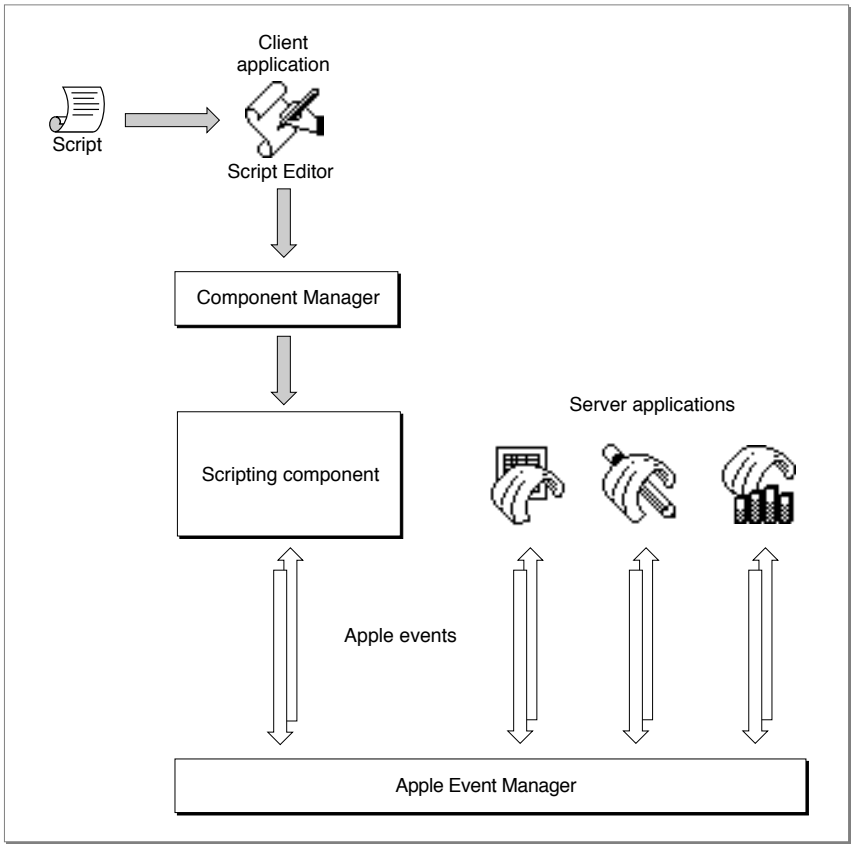
The OSA comprises the following parts:

- The Apple Event Manager allows applications to respond to Apple events sent by scripting components (see the previous section, "Sending and Responding to Apple Events").
- The *Apple Event Registry: Standard Suites* defines the standard vocabulary of Apple events.
- The standard scripting component data structures, routines, and resources allow applications to interact with any scripting component.
- The AppleScript component implements the AppleScript scripting language.

The ***AppleScript component,*** which implements the ***AppleScript scripting language,*** is the implementation of the OSA provided by Apple Computer, Inc. Users can view a script written in the AppleScript scripting language in several different ***dialects,*** or versions of the AppleScript language that resemble specific human languages or programming languages.

Figure 1-7 shows the relationships among some of these parts. The client application in Figure 1-7 is Script Editor, an application provided by Apple Computer, Inc., that allows users to record, edit, and execute scripts. The client application could also be any other application that uses the standard scripting component routines to execute scripts. Script Editor uses the Component Manager to open a connection with the scripting component that created the script to be executed.

Figure 1-7 How a scripting component executes a script



Like sound resources, scripts can be stored in applications and documents as well as in distinct script files that can be manipulated from the Finder. Script Editor allows users to execute scripts stored in ***script files.*** Users can also execute special script files called ***script applications*** simply by opening them from the Finder.

During script execution, scripting components perform actions described in the script, using the Apple Event Manager to send Apple events when necessary. The server applications shown in Figure 1-7 use the Apple Event Manager to examine the contents of the Apple events they receive and to respond appropriately. A server application always responds to the same Apple event in the same way, regardless of whether the event is sent by a scripting component or directly by a client application.

You can take advantage of the OSA in three ways:

- You can make your application ***scriptable,*** or capable of responding to Apple events sent to it by a scripting component. An application is scriptable if it
  - Responds to the appropriate standard Apple events. See the previous section, "Sending and Responding to Apple Events."
  - Provides an Apple event terminology extension (`'aete'`) resource that describes which Apple events your application supports and the corresponding human-language terminology for use in scripts. The `'aete'` resource allows scripting components to interpret scripts correctly and send the appropriate Apple events to your application during script execution.

  By executing scripts, users of scriptable applications can perform almost any task that they would otherwise perform by choosing menu commands, typing, and so on. Users can also execute scripts to perform many tasks that might otherwise be difficult to accomplish, especially repetitive or conditional tasks that involve multiple applications.
- You can make your application ***recordable*** — that is, capable of sending Apple events to itself in response to user actions such as choosing a menu command or changing the contents of a document. After a user has turned on recording for a particular scripting component, the scripting component receives copies of all subsequent Apple events and records them in the form of a script.
- You can have your application manipulate and execute scripts with the aid of a scripting component. To do so, your application must
  - use the Component Manager to open a connection with the appropriate component
  - use the standard scripting component routines to record, edit, compile, save, load, or execute scripts when necessary

  Users of applications that execute scripts can modify the applications' behavior by editing the scripts. For example, a user of an invoice program might be able to write a script that checks and if necessary updates customer information in a separate database application each time the user posts an invoice.

The sections that follow describe these three kinds of scripting capabilities in more detail. The chapter "Introduction to Scripting" in this book provides an overview of the way scripting components work and how you can implement support for scripting in your application.

Scriptable Applications

If your application can respond to standard Apple events sent by other applications, it can also respond to the same Apple events sent by a scripting component. Before executing a script that controls your application, a scripting component must associate the human-language terms used in the script with specific Apple event codes supported by your application. Scriptable applications provide this information in an Apple event terminology extension (`'aete'`) resource.

Because scripting components can obtain information from `'aete'` resources about the nature of different applications' support for Apple events, a single script can describe complex tasks performed cooperatively by several specialized applications. For example, a user can execute an AppleScript script to locate all records in a database with specific characteristics, update a series of charts based on those records, import the charts into a page-layout document, and send the document to a remote computer on the network via electronic mail.

When a user executes such a script, the AppleScript component attempts to perform the actions the script describes, including sending Apple events to various applications when necessary. To map human-language terms used in the script to the corresponding Apple events supported by each application, the AppleScript component looks up the terms in the applications' `'aete'` resources. Each human-language term specified by an application's `'aete'` resource has a corresponding Apple event code. After the AppleScript component has identified the Apple event codes for the terms used in a script, it can create and send the Apple events that perform the actions described in the script.

To respond appropriately to the Apple events sent to it by the AppleScript component, the database application in this example must be able to locate records with specific characteristics so that it can identify and return the requested data. The other applications involved must support Apple events that perform the other actions described in the script.

One line in such a script might be a statement like this:

```
copy Totals to chart "Summary of Sales" of document "Sales Chart"
```

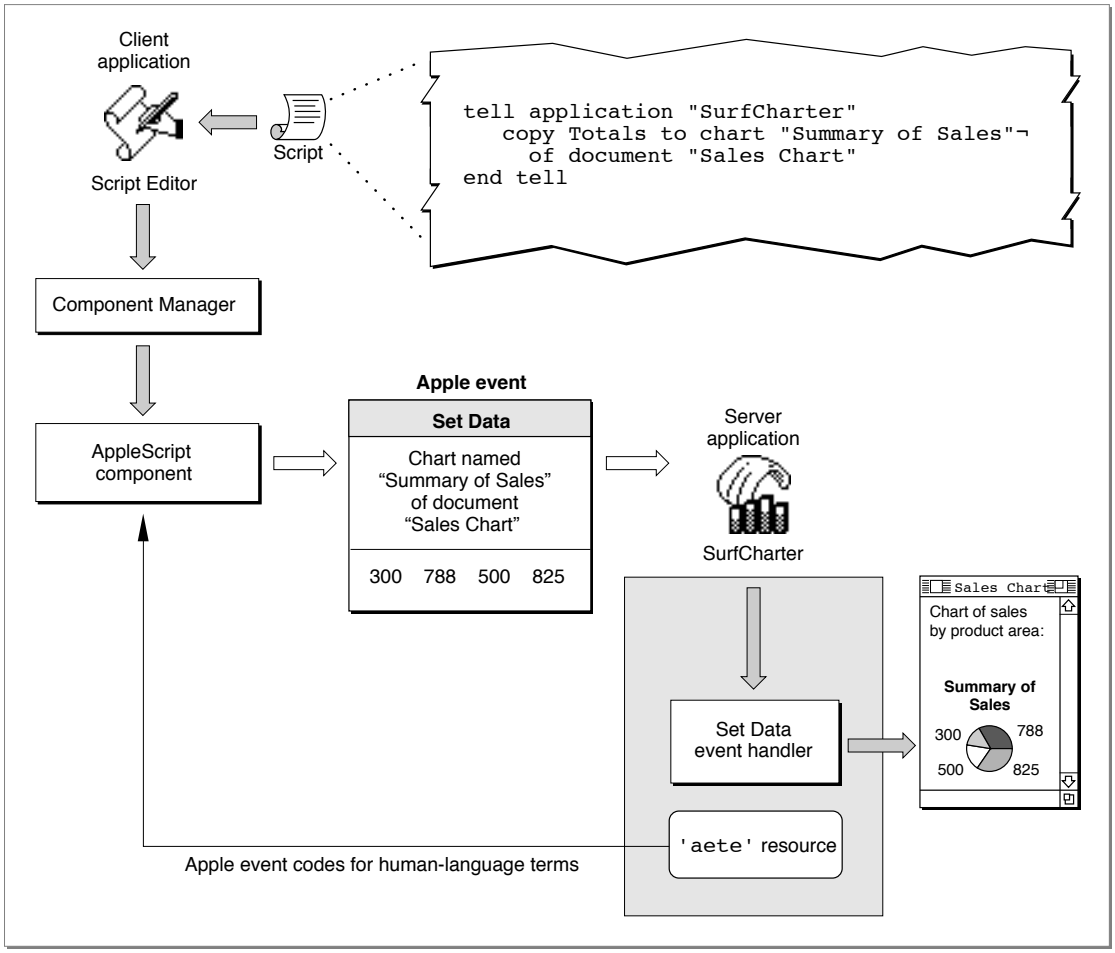
In this statement, the word `Totals` is a variable that has been set earlier in the same script to the value of the new data generated by a database application. The statement causes the AppleScript component to send a Set Data event updating the chart named "Summary of Sales." Figure 1-8 shows how the AppleScript component would execute this statement. (Figure 1-6 on page 1-12 shows a database application that sends a similar Set Data event directly.)

To interpret the terms in this script statement correctly, the AppleScript component must be able to look them up in the SurfCharter application's `'aete'` resource, which maps those terms to the corresponding codes for Apple events, object classes, and so on used by the Apple Event Manager. The AppleScript component can then create and send the Set Data event to SurfCharter.

When it receives the Set Data event, the SurfCharter application uses the Apple Event Manager to determine what kind of Apple event has been sent and to pass the event to SurfCharter's handler for that event, which in turn locates the chart and changes its data as requested.

The chapter "Introduction to Scripting" in this book describes how the `'aete'` resource works. The chapter "Apple Event Terminology Resources" describes how to define terminology for use by the AppleScript component and how to create an `'aete'` resource.

Figure 1-8 A Set Data event sent during script execution



Recordable Applications

If you decide to make your application scriptable, you can also make it recordable, allowing users to record their actions in your application in the form of a script. Even users with little or no knowledge of a particular scripting language can record their actions in recordable applications in the form of a script. More knowledgeable users can record scripts and then edit or combine them as desired.

Applications generally have two parts: the code that implements the application's user interface and the code that actually performs the work of the application when the user manipulates the interface. To make your application fully recordable, you should separate these two parts of your application, using Apple events to connect user actions with the work your application performs.

Any significant user action within a recordable application should generate Apple events that a scripting component can record as statements in a script. For example, when a user chooses New from the File menu, a recordable application sends itself a Create Element event, and the application's handler for that event creates the new document. Implementing Apple events in this manner is called ***factoring*** your application. A factored application acts as both the client and the server application for the Apple events it sends to itself.

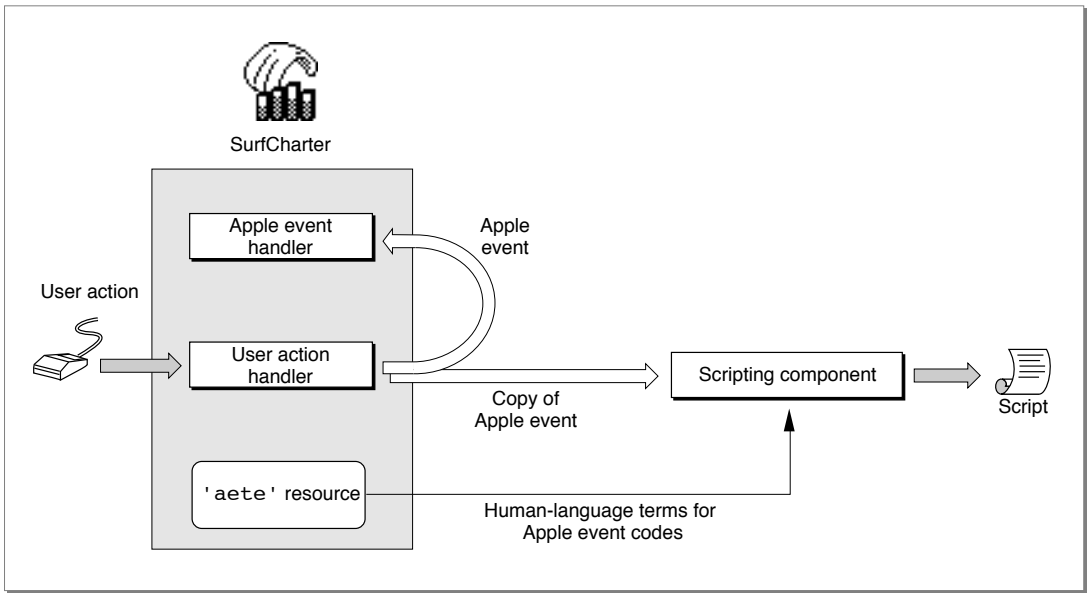
In general, a recordable application should generate Apple events for any user action that could be reversed by the Undo command. A recordable application can usually handle a greater variety of Apple events than it can record, since it must record the same action the same way every time even though Apple events might be able to trigger that action in several different ways.

A ***recordable event*** is any Apple event that any recordable application sends to itself while recording is turned on for the local computer (with the exception of events that the application indicates it does not want to be recorded). After a user turns on recording from the Script Editor application, the Apple Event Manager sends copies of all recordable events to Script Editor. A scripting component previously selected by the user handles each copied event for Script Editor by translating the event into the scripting component's scripting language and recording the translation as part of a Script Editor script. When a scripting component executes a recorded script, it sends the corresponding Apple events to the applications in which they were recorded.

Figure 1-9 illustrates how Apple event recording works. The user performs a significant action (such as choosing New from the File menu), and the SurfCharter application sends itself an Apple event to perform the task associated with that action. If recording is turned on, the Apple Event Manager automatically sends a copy of each recordable Apple event to the application (for example, Script Editor) that initiated recording. The scripting component handles the copy of each recordable event by translating it and recording it as part of a script. To translate each Apple event correctly, the scripting component must first check what equivalent human-language terminology the SurfCharter application uses for that Apple event. The scripting component then records the equivalent statement in the script.

The chapter "Recording Apple Events" in this book describes the Apple Event Manager's recording mechanism in more detail and explains how to use Apple events to factor your application.

Figure 1-9 Recording user actions in a factored application



Applications That Manipulate and Execute Scripts

Like sound resources, scripts can be stored either as separate files with their own icons in the Finder or within an application or its documents. Your application can store and execute scripts regardless of whether it is scriptable or recordable. If your application is scriptable, however, it can execute scripts that control its own behavior, thus acting as both the client application and the server application for the corresponding Apple events.

Your application can establish a connection with any scripting component that is registered with the Component Manager on the same computer. Each scripting component can manipulate and execute scripts written in the corresponding scripting language (or, as in the case of AppleScript, one of the scripting language's dialects) when your application calls the standard scripting component routines.

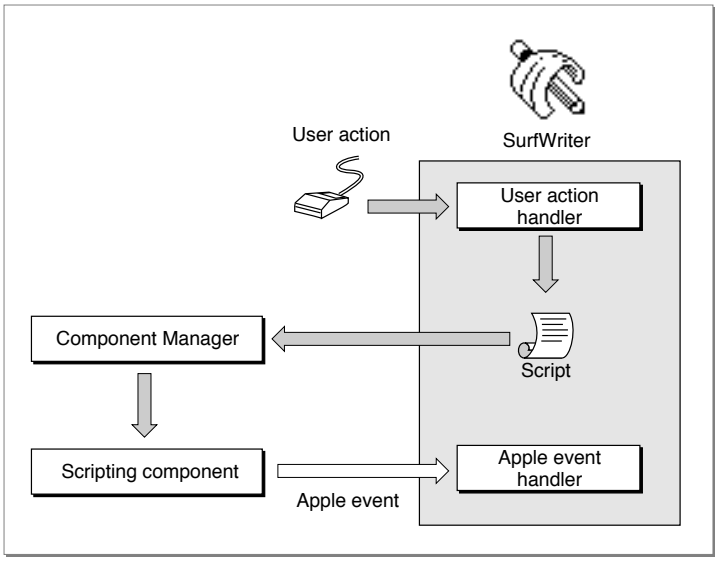
You can use the standard scripting component routines to

- get a handle to a script so you can save the script in a preferences file, in the data fork of a document, or as a separate script file
- manipulate scripts associated with any part of your application or its documents, including both Apple event objects and other objects defined by the application

- let users record and edit scripts
- compile and execute scripts

Figure 1-10 shows how an application might execute a script that controls its own behavior. The appropriate user action handler executes the script in response to a user action, which can be almost anything: choosing a menu command, clicking a button, tabbing from one table cell to another, and so on. The script might consist of a single statement that describes some default action, such as saving or printing, or a series of statements that describe a series of tasks, such as setting default preferences or styles. Figure 1-10 shows a script that corresponds to a single Apple event, but the script could just as easily correspond to a whole series of Apple events. If your application allows users to modify such a script, they can modify the behavior of your application to suit their needs.

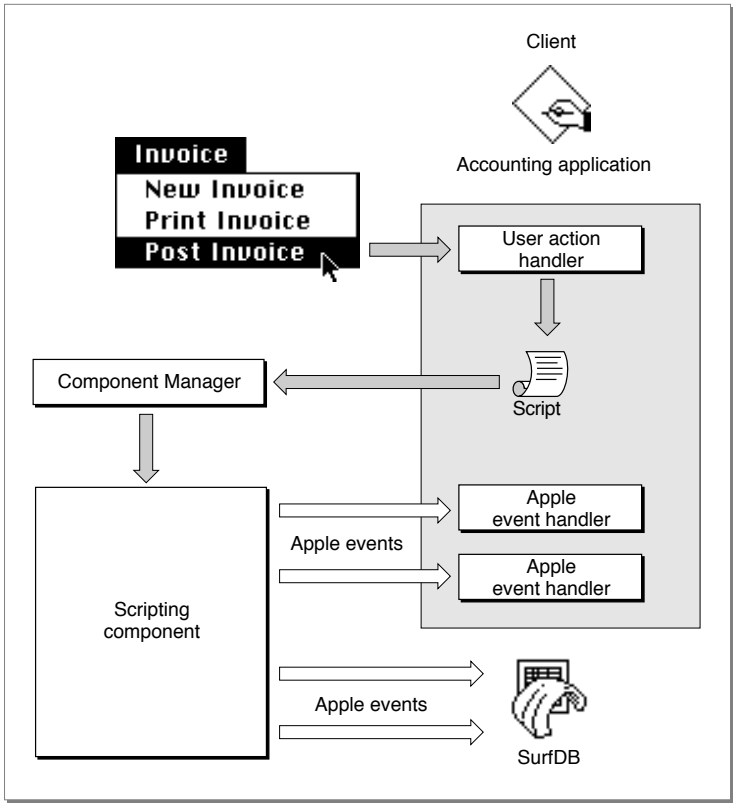
Figure 1-10 Controlling an application's own behavior by executing a script



Your application can associate a script with any Apple event object or application-defined object and execute the script when that object is manipulated in some way. The script can describe actions to be taken by your application, as in Figure 1-10, or actions to be taken by several applications. For example, a user of a word-processing application might attach a script to a specific word so that the application executes the script whenever that word is double-clicked. Such a script could trigger Apple events that look up and display related information from a separate document, run a QuickTime movie, perform a calculation, play a voice annotation, and so on.

Figure 1-11 shows one way that a script can be used to control two or more applications. When a user chooses the Post Invoice command in the accounting application, the user action handler for that menu command executes a default script for posting an invoice. That script might describe actions such as saving the invoice, updating the sales journal, and so on. The scripting component sends Apple events to the accounting application to perform these actions.

Figure 1-11 Posting an invoice and updating a database by executing a script



The accounting application also allows users to open the default invoice-posting script in Script Editor and modify it so that additional actions are performed when it is executed. For example, as shown in Figure 1-11, the script could instruct the SurfDB application to update a database of customer information in addition to performing the default posting actions. In this case, the scripting component sends Apple events to both the accounting application and SurfDB to carry out all the actions described by the script.

There is no limit to the actions such a script can describe. In addition to sending the Apple events shown in Figure 1-11, the invoice-posting script could be used to trigger Apple events that cause other applications to perform a credit check, send the invoice to the customer by electronic mail, forward inventory information to a remote server on the network, and so on.

The chapter "Scripting Components" in this book describes how your application can use the standard scripting component routines to manipulate and execute its own scripts and allow users to modify those scripts.

Exchanging Message Blocks

You should be able to meet most of your application's IAC needs by using the Apple Event Manager or the Event Manager. However, if you need low-level control or services not provided by the Apple Event Manager or the Event Manager, you can use the PPC Toolbox. The PPC Toolbox lets you send large amounts of data to other applications located on the same computer or across a network. The PPC Toolbox can also be used by pieces of code that are not event-driven. The PPC Toolbox is usually called by the Operating System; device drivers, desk accessories, or other code modules can also use it.

You cannot use the PPC Toolbox to send data between applications unless both your application and the application you're communicating with are open at the same time. To initiate communication, one program opens a port and requests a session with another program. The target application must also open a port and accept the request. Once a session is established, the two programs can read and write low-level message blocks.

The PPC Toolbox also provides a standard user interface that allows a user working in one application to select another application with which to exchange data, whether the communication is achieved by means of Apple events, other high-level events, or message blocks.

The chapter "Program-to-Program Communications Toolbox" in this book describes how programs can exchange low-level message blocks.