Requesting Services Through Apple Events

Your application can use Apple events to request services from other applications. By using Finder events, for example, your application can simulate the behavior of the Finder by requesting that the Finder perform such operations as launching an application on your behalf.
Launch Application with Doc Using Apple Events is a code example which shows how to accomplish this. By using functional-area Apple events, your application can request services from applications related to your own-for example, asking a spelling checker application to check the text in a document created by your application. All publicly available Apple events are defined and published in the *Apple Event Registry.*   Consult the *Apple Event Registry* for the format and function of Apple events that your application may wish to send.

The steps your application must take to act as a client application and request such services are described below. To request a service through an Apple event, your application must:

- create an Apple event by calling the **AECreateAppleEvent** function
- use **Apple Event Manager** functions to add parameters and any other necessary attributes to the Apple event
- call the **AESend** function to send the Apple event
- dispose of any copies of descriptor records that you have created
- process the reply Apple event (optional)

Use the **AECreateAppleEvent** function to create an Apple event record. Using the arguments you pass to the **AECreateAppleEvent** function, the **Apple Event Manager** constructs the data structures describing the event class, the event ID, and the target address attributes of an Apple event. The event class and event ID, of course, identify the particular event you wish to send. The target address identifies the application to which you wish to send the Apple event.

To act as a server application for your application, the target must support high-level events and must be open. The server can be your own application, another application running on the user's computer, or an application running on another user's computer connected to the network. Your application should offer some facility to launch a server application if it is not already running. It is recommended that you use the
Open Selection event (identified by the event class kAEFinderEvents and the event ID kAEOpenSelection) to request that the Finder launch applications; **Launch Application with Doc Using Apple Events** is a code example which shows how to accomplish this. However, the **Process Manager** also provides a means for your application to launch other applications. See the *Apple Event Registry* for more information on Finder events.

Your application should also offer a facility to allow the user to choose among the various applications available as servers. The **PPCBrowser** function allows users to select target applications on the user's computer as well as those available on computers connected to the network. The **PPCBrowser** function presents a standard user interface for choosing a target application, much as the **Standard File Package** provides a standard user interface for opening and saving files. See the description of **PPC Toolbox** for details on using the **PPCBrowser** function.

If the server application is on a remote computer on a network, the user of that computer must allow program linking to the server application. The user of the server application does this by selecting the application from the Finder and choosing **Sharing** from the **File** menu and then clicking the **Allow Remote Program Linking** check box. If the user has not yet started program linking, the **Sharing** command offers to display the **Sharing Setup** control panel so that the user can start program linking. The user must also authorize remote users for program linking by using the **Users and Groups** control panel. Program linking and setting up authenticated sessions are described in **PPC Toolbox**.

There are two other attributes you specify in the **AECreateAppleEvent** function: the reply ID and the transaction ID. For the reply ID attribute, you'll usually specify the kAutoGenerateReturnID constant to the **AECreateAppleEvent** function. This constant ensures that the **Apple Event Manager** generates a unique return ID for the reply Apple event returned from the server. For the transaction ID attribute, you'll usually specify the kAnyTransactionID constant, which indicates that this Apple event is not one of a series of interdependent Apple events.

The Apple event record created with the **AECreateAppleEvent** function serves as a template for the Apple event you want to send. To add the remaining attributes and parameters necessary for your Apple event, you must use these additional **Apple Event Manager** functions:, **AEPutParamPtr**, **AEPutParamDesc**, **AEPutAttributePtr**, and **AEPutAttributeDesc**.

Descriptor records and discriptor lists are the basic components from which an Apple event record is constructed; these are passed to **AEPutParamDesc** and **AEPutAttributeDesc** functions. Use the following functions to create descriptor records and descriptor lists: **AECreateDesc**, **AEPutPtr**, and **AEPutDesc**.

After you add all the attributes and parameters required for the Apple event, use the **AESend** function to send the Apple event. The **Apple Event Manager** uses the **Event Manager** to transmit the Apple event to the server application.

The **AESend** function requires that you specify whether and how your application should wait for a reply from the server. When the server receives your Apple event, the **Apple Event Manager** prepares a reply Apple event for your application by passing a default reply Apple event to the server. The **Apple Event Manager** returns any nonzero result code from the server's handler in the keyErrorNumber parameter of the reply Apple event. If your application wants to return an error string, add it to the reply Apple event in the keyErrorString parameter. The server can also use this reply Apple event to return any data you requested-for example, the results of a numerical calculation or a list of misspelled words.

After you send an Apple event, your application is responsible for disposing of the Apple event record-and thereby deallocating the memory it uses-by calling the **AEDisposeDesc** function. If you create one descriptor record and add it to another, the **Apple Event Manager** creates a copy of the newly created one and adds that copy to the existing one. For example, you might use the **AECreateDesc** function to create a descriptor record that you wish to add to an Apple event. When you use the **AEPutParamDesc** function, it creates a copy of your newly created descriptor record and adds that copy as a parameter to an existing Apple event.

Your application should dispose of all the descriptor records that are created in order to add parameters and attributes to an Apple event. You normally dispose of your Apple event and its reply after you receive a result from the **AESend** function. You should dispose of these even if **AESend** returns an error result. If your application requests a reply Apple event, your application must also dispose of it when finished processing it.

Your application can request a reply Apple event. If you specify the kAEWaitReply flag, the reply Apple event is returned in a parameter you pass to the **AESend** function. If you specify the kAEQueueReply flag to the **AESend** function, the reply Apple event is returned in the event queue. In this case, the reply is identified by the event class kCoreEventClass and the event ID kAEAnswer; your application processes reply events that it receives in its event queue in the same manner that server applications process Apple events

Your application should check for the keyErrorNumber parameter of the reply Apple event to ensure that the server performed the requested action. Any error messages that the server returns for you to display to your user will appear in the keyErrorString parameter.

When your handler is finished using a copy of a descriptor record used in the reply Apple event, you should dispose of them both-and thereby deallocate the memory they use-by calling the **AEDisposeDesc** function.