Responding to Apple Events

A client application uses the **Apple Event Manager** to create and send an Apple event requesting a service. A server application responds by using the **Apple Event Manager** to process the Apple event, to extract data from the attributes and parameters of the Apple event, and to return a result to the client application. The server provides its own routines for performing the action requested by the client's Apple event.

As its first step in supporting Apple events, your application must be able to respond to the required Apple Events sent by the Finder. If you plan to implement publish and subscribe capabilities, your application must respond to the Apple events sent by the **Edition Manager**. You can also respond to Apple events sent by your own application or by other applications. This section provides a quick overview of the steps your application takes in responding to Apple events.

To respond to Apple events, your application must

- test for high-level events in its event loop
- use the **AEProcessAppleEvent** function to process Apple events
- provide handler routines for the Apple events it supports
- use **Apple Event Manager** functions to extract the parameters and attributes from Apple events
- use the **AEInteractWithUser** function-if your application requires input from the user when your application is responding to an Apple event-to bring your application to the foreground to interact with the user
- return a result for the client

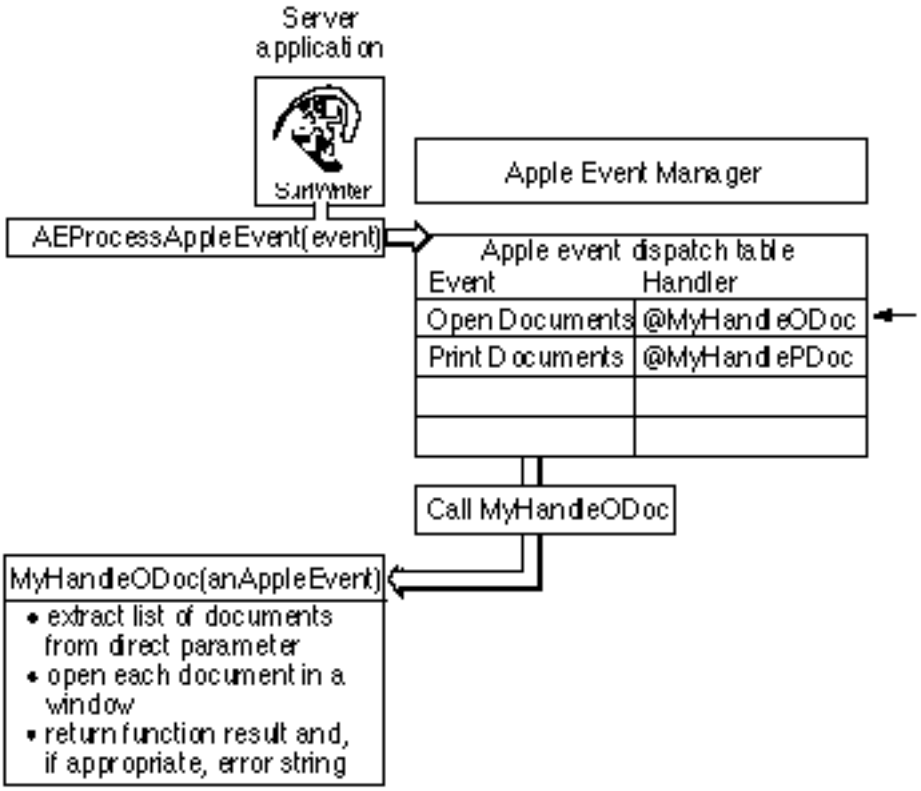
Note that in order for your application to respond to Apple events sent from remote computers, the user of your application must allow network users to link to your application. The user does this by selecting your application from the Finder and choosing **Sharing** from the **File** menu and then clicking 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 covered in the **PPC Toolbox** description.

An Apple event (like all high-level events) is identified by a message class of kHighLevelEvent in the what field of the EventRecord. You test the what field of the event record to determine whether an event is a high-level event. If the what field contains the kHighLevelEvent constant and your application defines any high-level events other than Apple events, test the message field of the event record to determine whether the high-level event is something other than an Apple event. If the high-level event is not one that you've defined for your application, assume that it is an Apple event. (Note that you are encouraged to use Apple events instead of defining your own high-level events whenever possible.)

After determining that an event is an Apple event, use the **AEProcessAppleEvent** function to let the **Apple Event Manager** identify the event.

The **AEProcessAppleEvent** function begins processing the Apple event. The **AEProcessAppleEvent** function identifies the Apple event by examining the data in the event class and event ID attributes. The **AEProcessAppleEvent** function in turn uses that data to call the Apple event handler that your application provides for that event. An Apple event handler is a function that extracts the pertinent data from the Apple event, performs the action requested by the Apple event, and returns a result. For example, if the event has an event class of kCoreEventClass and an event ID of kAEOpenDocuments, the **AEProcessAppleEvent** function calls your application's routine for handling the Open Documents event.

You install Apple event handlers by using the **AEInstallEventHandler** function. This function creates an Apple event dispatch table that the **Apple Event Manager** uses to map Apple events to handlers in your application. After being called by the **AEProcessAppleEvent** function to process an Apple event, the **Apple Event Manager** reads the Apple event dispatch table and, if your application has installed a handler for that Apple event, calls your handler to finish responding to the event. The following picture shows how the flow of control passes from your application to the **Apple Event Manager** and back to your application.



The **Apple Event Manager** calling the handler for an Open Documents event

Your Apple event handlers must generally perform the following tasks:

- extract the parameters and attributes for the Apple event

- check that all the required parameters have been extracted
- set user interaction level preferences if necessary and, if your application needs to interact with the user, use the **AEInteractWithUser** function to bring it to the foreground
- perform the action requested by the Apple event
- dispose of any copies of descriptor records that have been created
- return a result for the client

You must use **Apple Event Manager** functions to extract the data from Apple events. You can also use **Apple Event Manager** functions to get data out of descriptor records, descriptor lists, and AE records. Most of these routines are available in two forms: one that uses a buffer to return a copy of the desired data, and one that returns a copy of the descriptor record containing the data. These functions are **AEGetParamPtr**, **AEGetParamDesc**, **AEGetAttributePtr**, **AEGetAttributeDesc**, **AECountItems**, **AEGetNthPtr**, **AEGetNthDesc**. See **Getting Data Out of an Apple Event** for more information on using these functions.

After extracting all known parameters, your handler should check that it retrieved all the required parameters by checking whether the keyMissedKeywordAttr attribute exists. If the attribute exists, then your handler has not retrieved all the required parameters, and it should return an error. See **Writing Apple Event Handlers** for a code example that checks that all required parameters have been retrieved.

In some cases, the server may need to interact with the user when it handles an Apple event. For example, your handler for the Print Documents event may need to display a print options dialog box and get settings from the user before printing. Your handler should always use the **AEInteractWithUser** function before displaying a dialog box or alert box or otherwise interacting with the user. By specifying a flag indicating the level of user interaction allowed, to the **AESetInteractionAllowed** function, you can set your application's user interaction level preferences. See **Interacting With the User** for more information on setting up user interaction as the result of an Apple Event.

When your application acts on an Apple event, it should perform the standard action requested by that event. For example, if the Apple event is the Open Documents event, your application should open the specified documents in titled windows just as if the user had selected each document from the Finder and then chosen Open from the File menu. You should strive to create routines that can be called in response to both user events and Apple events. To do this, you need to isolate code for interacting with the user from the code that performs the requested action-such as opening a document. You then call the code that performs the requested action from your Apple event handler.

When you extract a descriptor record by using the **AEGetParamDesc**, **AEGetAttributeDesc**, **AEGetNthDesc**, or **AEGetKeyDesc** function, the **Apple Event Manager** creates a copy of the descriptor record for you to use. When your handler is finished using a copy of a descriptor record, you should dispose of it-and thereby deallocate the memory it uses-by calling the

AEDisposeDesc function. See **Disposing of Apple Event Data Structures** for more information on when to dispose of descriptor records.

The required Apple Events ask your application to perform tasks-open your application, open or print documents, or quit your application. Other Apple events may ask your application to return data. For example, if your application is a spelling checker, the client probably expects data in the form of a list of misspelled words to be returned from your application. If a reply is requested, the **Apple Event Manager** prepares a reply Apple event for the client by passing a default reply Apple event to your handler. The default reply Apple event has no parameters when it is passed to your handler. Your handler can add any parameters to the reply Apple event. If your application is a spelling checker, for example, you can return a list of misspelled words in a parameter.

Your handler routine should always set its function result either to noErr if it successfully handles the Apple event or to a nonzero result code if an error occurs. If an error occurs, the **Apple Event Manager** adds a keyErrorNumber parameter to the reply Apple event; this parameter contains the result code that your handler returns. The client should check whether the keyErrorNumber parameter exists to determine whether your handler performed the requested action. In addition to returning a result code, your handler can also return an error string in the keyErrorString parameter of the reply Apple event. The client can use this string in an error message to the user.If the source requested a reply, the **Apple Event Manager** returns the reply Apple event to the source. The reply Apple event is identified by the event class kCoreEventClass and by the kAEAnswer. When you have finished using the reply Apple event, you should dispose of it-and thereby deallocate the memory it uses-by calling the **AEDisposeDesc** function. See **Replying to an Apple Event** for a code example showing how to add the keyErrorString parameter to the reply Apple Event.