Accepting an Apple Event

To accept Apple events (or any other high-level events), you must set the appropriate flags in your application's 'SIZE' resource and include code to handle high-level events in your application's main event loop.

Two flags in the 'SIZE' resource determine whether an application receives high-level events:

- The isHighLevelEventAware flag must be set for your application to receive any high-level events.
- The localAndRemoteHLEvents flag must be set for your application to receive high-level events sent from another computer on the network.

Note that in order for your application to respond to Apple events sent from remote computers, the user of your application must also 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 described in the **PPC Toolbox** description.

Apple events (and other high-level events) are identified by a message class of kHighLevelEvent in the what field of the event record. You can test the what field of the event record to determine whether the event is a high-level event.

The following program is an example of a procedure called from an application's main event loop that handles events, including high-level events. The procedure determines the event type received and then calls another routine to take the appropriate action.

```
// A DoEvent function

// Assuming inclusion of <MacHeaders>

#include<AppleEvents.h>

void DoMouseDown (EventRecord *event);
void DoHighLevelEvent (EventRecord *event);
void DoEvent(EventRecord *event);

void    DoEvent(EventRecord *event)
{
    switch (event->what) {
    case mouseDown :
        DoMouseDown(event);

    // handle other kinds of events here
```

```
        // handle high-level events, including Apple events
        case kHighLevelEvent :
            DoHighLevelEvent(event);
        }
    }
```

The following program is an example of a **DoHighLevelEvent** procedure that handles Apple events and also handles the high-level event identified by the event class mySpecialHLEventClass and the eventID *mySpecialHLEventID*. Note that, in most cases, you should use Apple events to communicate with other applications.

```
// A DoHighLevelEvent function for handling
// Apple events and other high-level events.

// Assuming inclusion of <MacHeaders>

#include <AppleEvents.h>
OSErr HandleMySpecialHLEvent (EventRecord *event);
void     DoError (OSErr  myErr);
void     DoHighLevelEvent (EventRecord *event);

void     DoHighLevelEvent(EventRecord *event)
{
    OSErr  myErr;

    if ((event->message == mySpecialHLEventClass) &&
        ((*((long *) (&(event->where)))) ==
                (long)mySpecialHLEventID))
    {
        // it's a high-level event that doesn't use AEIMP
        myErr = HandleMySpecialHLEvent(event);
        if (myErr)
            DoError(myErr);                 // perform the necessary
    }                                       // error handling
    else {
        // otherwise, assume that the event is an Apple event
        myErr = AEProcessAppleEvent(event);
        if (myErr)
            DoError(myErr);
    }
}
```

If your application accepts high-level events that do not follow the Apple Event Interprocess Messaging Protocol (AEIMP), you must dispatch these high-level events before calling **AEProcessAppleEvent**. To dispatch a high-level event that does not follow AEIMP, for each event you should check the event class, the event ID, or both to see if the event is one that your application can handle.

After receiving a high-level event (and, if appropriate, checking whether it is a type of high-level event other than an Apple event), your application typically calls the **AEProcessAppleEvent** function. The **AEProcessAppleEvent** function determines the Apple event type received, gets the event buffer that contains the parameters and attributes of the Apple event, and calls the corresponding Apple event handler routine in your application.

You should provide an Apple event handler routine for each Apple event that your application supports. Your handler routine for a particular Apple event is responsible for performing the action requested by the event, and your handler can optionally return data in the reply Apple event.

After your handler finishes processing the Apple event and adds any parameters to the default reply, it should return a result code to AEProcessAppleEvent. If the client application is waiting for a reply, the **Apple Event Manager** returns the reply Apple event to the client.