Sending High-Level Events

You use the **PostHighLevelEvent** routine to send a high-level event to another application. When doing so, you need to provide six pieces of information:

- an EventRecord with the EventClass and eventID assigned appropriately
- the identity of the recipient of the event
- a unique number that identifies this particular event
- a data buffer that can contain optional data
- the length of the data buffer
- options determining how the event is posted

Note: To send an Apple event, use the **Apple Event Manager** function **AESend**. It uses the **Event Manager** to post Apple events.

As indicated in the previous section, you can identify the recipient of the high-level event in one of four ways. the following program illustrates how to send a high-level event to an application on the local machine. In this example, an application is sending an event to an application whose signature is 'boff'.

Posting a high-level event by application signature

```
// Assumes inclusion of <MacHeaders>

#include <EPPC.h>

void PostTest (void);
void DoError (OSErr myErr);
OSErr PostWithPPCBrowser (void);

void PostTest ()
{
    EventRecord    myEvent;          // An event record
    OSType myRecvID;// ReceiverID
    long     myOpts;                 // Posting options
    OSErr  myErr;

    myEvent.what  = kHighLevelEvent;
    myEvent.message  = (long) 'boff';
    myEvent.where.h  = 'cm';
    myEvent.where.v  = 'd1';

    myOpts     = receiverIDisSignature + nReturnReceipt;
    myRecvID  = 'boff';
    myErr      = PostHighLevelEvent( &myEvent, myRecvID, 0, nil, 0,
                                     myOpts);
    if (myErr)
        DoError(myErr);
```

```
}
```

In this example, there is no additional data to transmit, so the sending application provides NIL as the pointer to the data buffer and sets the buffer length to 0. Note that the receiver is specified by its creator signature and that the sender requests a return receipt. The myOpts parameter specifies posting options, which are of two types: delivery options and options associated with the receiverID parameter. You can specify one or more delivery options to indicate if you want the other application to receive the event at the next opportunity and to indicate if you want acknowledgment that the other application received the event. You use the options associated with the receiverID parameter to indicate how you are specifying the recipient of the event. To set the various posting options, use constants.

When you specify the receiver of the event by targetID, use the constant receiverIDisTargetID in the postingOptions parameter and specify a pointer to a TargetID record for the receiverID parameter.

When you pass a targetID record, you need to specify only the name and location fields. You can use the **IPCListPorts** function to list all of the existing port names along with information on whether the port will accept authenticated service on the machine specified by the port location name. For information on how to use the **IPCListPorts** function, see the **PPC Toolbox** description.

You can also use the **PPCBrowser** function to fill in a targetID record. The following program illustrates how to use the **PPCBrowser** function to post a high-level event. In this example, the sending application wants to locate a dictionary application and have the dictionary return the definition of a word to it.

Using the PPCBrowser function to post a high-level event

```
#include <EPPC.h>

Ptr gATextPtr;
unsigned long gTextLength;

OSErr PostWithPPCBrowser ()
{
    EventRecord myHLEvent;
    OSErr myErr;
    short myNumTries;
    PortInfoRec myPortInfo;
    TargetID myTarget;

    // Use PPCBrowser to get the target
    myErr = PPCBrowser((ConstStr255Param) "\pSelect an application",
                (ConstStr255Param) "\pApplication",
                FALSE, &myTarget.location, &myPortInfo,
                nil, (ConstStr32Param) "\p");
    if (!myErr)
    {
```

```
        // Copy portname into myTargetName
        myTarget.name  = myPortInfo.name;
        myHLEvent.what = kHighLevelEvent;
        myHLEvent.message = (long) 'DICT';
        myHLEvent.where.v ='De';
        myHLEvent.where.h = 'fn';

        // If a connection is broken, then sessClosedErr is returned to
        // PostHighLevelEvent;  to reestablish the connection, just post
        // the event one more time

        myNumTries = 0;

        do
        {
            myErr = PostHighLevelEvent(&myHLEvent,(long) &myTarget,
                        0, gATextPtr, gTextLength, receiverIDisTargetID);
            myNumTries++;
        } while ((myErr == sessClosedErr) || (myNumTries <= 1));
    }
    return myErr;
}
```

This example puts up a dialog box asking the user to select a dictionary. When one is selected, this code posts a high-level event to that dictionary application asking for the definition of the selected text. Note that the sending application *and* the receiving application must both agree that definition queries are to be of event class 'Dict' and eventID 'Defn'. It is necessary to define a private protocol only in cases where no suitable
Apple event exists.

> **Note:** You should avoid passing handles to the receiving application in an attempt to share a block of data. It is better to put the relevant data into a buffer (as illustrated in the previous program) and pass the address of the buffer. If you absolutely must share data by passing a handle, make sure that the block of data is located in the system heap.

If a high-level event is posted successfully, **PostHighLevelEvent** returns the result code noErr, which indicates only that the event was successfully passed to the **PPC Toolbox**. Your application needs to call another **Event Manager** routine (**EventAvail**, **GetNextEvent** or **WaitNextEvent**) to give the other application an opportunity to receive the event.

The event you send may require the other application to return some information to your application by sending a high-level event back to your application. You can scan for the response by using the function **GetSpecificHighLevelEvent**. If your application must wait for this event, you might want to display a watch cursor or take other action as appropriate to your application. You also might want to implement a timeout mechanism in case your application never receives a response to the event.