Apple Events 101

UserLand Software, Inc.

Synopsis

Basic introduction to Apple Events programming using THINK C. Two sample programs in source code: Client and Server. Client sends a series of Apple Events to Server. They are minimal programs, makes it easy to see how the Apple Events work. Requires THINK C 5.0 or greater. Readme in Word 4.0 format. Packed with StuffIt 3.0. Does not require Frontier or Frontier Runtime.

Requires System 7.0, Universal Headers, MacsBug/TMON

The Apple Event Manager, which this sample code builds on, first appeared in System 7.0. These programs will not run on earlier versions of the Macintosh operating system.

The source code samples were written using Think C 7.0 and Metrowerks Code Warrior 1.0, but they don't rely on features that are only present in these versions. You can build this code with any compiler compatible with Apple's Universal Headers.

Error reporting is done via DebugStr calls. So you should have MacsBug or TMON loaded to report errors without crashing your system.

A pair of sample programs

There are two programs in this package: Client and Server. They illustrate the basic techniques of Macintosh interapplication communication using the Apple Event Manager.

Server has a standard Apple menu. Its File menu contains a single command, Quit. It installs handlers for the four required Apple Events and a handler for a custom event that's called by the Client app.

Client is even simpler than Server. It has no menus and no windows. It initializes the Macintosh managers and then drops into a loop. Each time thru the loop it sends an Apple Event asking Server to display a number between 1 and 100. It checks errors, reporting them via DebugStr.

These programs are minimal in every way. That makes it easy to find the Apple Event code.

You may copy the source code from these applications into any other program without any royalty or license fee. You may redistribute this package exactly as-is. You may not distribute a modified version of this package.

A pair of experiments

Experiment #1: Be sure the programs work on your machine.

1. Be sure MacsBug or TMON is loaded.

2. Launch Server.

3. Launch Client.

You should see a count from 1 to 100 in the Server window. Then Client quits. Server doesn't quit.

Experiment #2: See how errors are reported

1. Be sure MacsBug or TMON is loaded.

2. Launch Server.

3. Launch Client.

4. Quickly bring Server to the front.

5. Quit.

Client should halt with an error code of -609, an Event Manager error which according to Think Reference means "invalid connection".

Certainly you have more luxurious error reporting in <u>your</u> program. ;-)

Touring the source code

1. **Open server.π, the project file for the Server app. Open server.c. Server sits in its main event loop, like all Macintosh applications, waiting for an interactive event, such as a mouse click or keystroke. Also, in its main event loop, it watches for incoming Apple Events and uses AEProcessAppleEvent to dispatch to its Apple Event handler routines.**

2. **Search for handleEvent. All the code above this routine deals with the usual things that Macintosh programs have to deal with.**

3. **Look at the switch statement in handleEvent. There's a branch for kHighLevelEvent, and in that branch is a call to AEProcessAppleEvent. If your program is going to handle incoming Apple Events, you must call AEProcessAppleEvent on receipt of a HighLevelEvent.**

4. **AEReturnError returns an error to the caller. You may want to copy it directly into your program to make it easy to report errors on processing incoming Apple Events.**

5. **openDocEvent and printEvent don't do anything since this program doesn't open files or print them.**

6. **How setMessageEvent works**

    The setMessageEvent Apple Event handler expects a single parameter whose key is '----', and type is 'TEXT', containing a handle to the text to be displayed in its window.

    If the handle contains more than 255 characters, only the first 255 characters are copied.

    If the '----' parameter isn't present, setMessageEvent returns an error to the caller, following the error reporting conventions supported by Frontier and AppleScript.

    If everything goes well, it copies the text into the windowmessage string and invalidates the window.

    To illustrate how returned values work, setMessageEvent converts the string to a number, and adds it to the Apple Event reply. The Client app checks for this reply.

7. **quitEvent sets the flexitmainloop boolean true, which causes the program to exit from its main event loop. It's essential that the quit handler return normally to the Apple Event Manager to avoid hanging the system.**

8. **How installHandlers works**

   installHandlers attempts to install five Apple Event handlers.

   Each handler is installed with a four-character class and an identifier. In the first call to AEInstallEventHandler, we associate the class 'SERV' and id 'DISP' with the setMessageEvent routine. The last parameter, false, says this is not a system event handler, the second-to-last parameter, 0, indicates that this handler doesn't have a refcon.

   When an Apple Event whose class is 'SERV' and id is 'DISP' arrives for this application, AEProcessAppleEvent will call setMessageEvent.

   The AppleEvents.h include file contains #defines for the standard classes and ids used in the subsequent AEInstallEventHandler calls.

9. **Check out the SIZE resource**

   Choose the Set Project Type command in Think C's Project menu. The SIZE setting is 58E0, indicating that the following flags are on:

   ```
   Multifinder-aware

   Background null events

   Suspend & resume events

   32-bit compatible

   HighLevelEvent-aware

   Accept remote HighLevelEvents
   ```

   The major one, clearly, is HighLevelEvent-aware, although we think the others generally follow -- if you're scriptable, you should be scriptable over the network, and you very likely need background null events and suspend and resume events.

10. **Close server.π and open client.π. Open client.c.**

11. **Starting at the top, initMacintosh initializes the usual Macintosh   managers.**

12. **Jump to the end of the file and look at the main function. It loops 100 times, sending an Apple Event to Server each time through the loop.**

13. **How sendMessageToServer works**

    It creates an Apple Event, converts the number to a string, sends the event to Server, waits for a reply, and checks the reply to be sure everything worked.

    It's a complicated routine, so we broke it up into four sections. The following comments apply to each of the sections:

    1. **Creates an address descriptor using Server's creator id. We use that address to create an Apple Event by calling AECreateAppleEvent.**

    2. **Converts the number to a string using NumToString, and then adds it onto the Apple Event.**

    3. **Sends the message and waits for a reply using AESend.**

    4. **Finally, we look at the reply, and check to see that it's a long, containing the number we started with.**

    If any errors occur, we use DebugStr to report them. If you see no crashes running this program, you know that all is well!

    Finally, we dispose of the address descriptor, the event we create and the reply. Note that all are initialized to {typeNull, nil} at the beginning of the routine, so that the calls to AEDisposeDesc will work even if an error causes the event not to be created or sent.

14. **Set Project Type**

    As with the Server app, check the Set Project Type command. We've set the SIZE resource to 58E0, the same as Server.

15. **Client could be a script**

    The Client app is playing the role that's usually played by a Frontier or AppleScript script.

    Here's what the Frontier script would look like:

    ```
    local (i)
    for i = 1 to 100
            appleEvent ('AESV', 'SERV', 'DISP', '----', string (i))
    ```

    The details of the appleEvent call could be hidden in a subroutine:

    ```
    on display (num)
            return (appleEvent ('AESV', 'SERV', 'DISP', '----', string (num)))
    local (i)
    for i = 1 to 100
            display (i)
    ```

    We've included a copy of this script in clientAsScript in the Client folder. You can run it with Frontier or Frontier Runtime 2.0.

Apple Events 102 and beyond

That wraps it up for Apple Events 101. With this sample source code, you should be able to add a set of Apple Event handlers to any main-event-loop based Macintosh program.

We have ideas for what would make a good AE 102 upload. For example, we're working on a minimal document-based application that implements a set of Apple Events that open and close windows, save documents, let scripts move data into the application and get data from the application. Again, the goal will be to be supply an absolutely minimal application, to make it easy for you to borrow code to include in your software.

What would you like to see in future Apple Events sample code?

Please get in touch thru Section/Library 16 at GO USERLAND on CompuServe. Or thru our AppleLink and America On-Line forums. Look for more sample C source code, documentation and sample scripts in these places.