10

APPLE EVENTS

Demonstration Program: AppleEvents

Introduction

Apple events are **high-level events** whose structure and interpretation are determined by the **Apple Event InterProcess Messaging Protocol** (AEIMP). Applications typically use Apple events to request services and information from other applications and to provide services and information in response to such requests.

In the world of Apple events:

- The **client application** is the application that initiates communication between two applications that support Apple events. It sends the Apple event that requests a service or information.
- The **server application** is the application that provides the requested service or information.

As an example, Fig 1 shows a client application (the Finder) sending an Apple event known as the Open Documents event to the server application (My Application) requesting the latter to open the documents named Document A and Document B. My Application responds by opening windows for the specified documents.

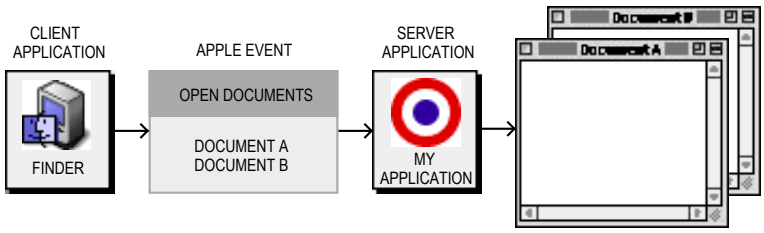

FIG 1 - CLIENT AND SERVER

An application can also send Apple events to itself, thus acting as both client and server.

Applications can rely on a vocabulary of standard Apple events, defined in the Apple Event Registry: Standard Suites, to identify Apple events and respond appropriately. The standard **suites** (a suite being a group of Apple events that are usually implemented together) include:

- The **required** suite, which consists of five Apple events that the Finder sends to applications. The **required Apple events** are:
  - Open Application.
  - Re-open Application.
  - Open Documents.

- Print Documents.
- Quit Application.

The Finder uses the required Apple events as part of the mechanism for launching and terminating applications. Your application must support the required Apple events.

- The **core** suite, which consists of the basic Apple events, including Get Data, Set Data, Move, Delete and Save, that nearly all applications use to communicate.
- The **functional-area** suite, which consists of a group of Apple events that support a related functional area, and which include the Text suite and the Database suite.
- The **appearance** suite, which pertains to Mac OS 8/9 only, and which consists of four Apple events used to advise all foreground applications when one of the following changes has been made in the Mac OS 8/9 Appearance control panel:
  - Appearance. (Since the only Appearance ever made available by Apple is Platinum, this event will not be considered further.)
  - Large system font.
  - Small system font.
  - Views font.

Another important Apple event, of relevance on Mac OS X only, is the Show Preferences event. This event is sent to your application when the user chooses the **Preferences...** item in the Mac OS X Application menu.

This chapter is primarily concerned with the required Apple events, the Appearance Manager Apple events, and the Show Preferences event ,exploring the subject of Apple events only to the extent necessary to gain an understanding of the measures involved in supporting these events.

Apple Event Attributes and Parameters

An Apple event comprises **attributes** and, possibly, **parameters.** Attributes identify the Apple event and denote the task to be performed. Parameters contain information to be used by the target application.

Apple Event Attributes

An Apple event attribute is a structure which identifies, principally, the **event class**, **event ID**, and target application.

Event Class

The event class identifies a group of related Apple events. It appears in the `message` field of the event structure for an Apple event (see Fig 2). For example:

- The required Apple events have the value `'aevt'` in the `message` field of their event structures. (`'aevt'` is represented by the constant `kCoreEventClass`.)
- The Appearance Manager Apple events have the value `'appr'` in the `message` field of their event structures. (`'appr'` is represented by the constant `kAppearanceEventClass`.)

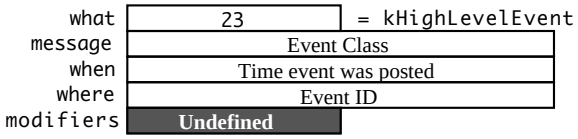

FIG 2 - CONTENTS OF AN EVENT STRUCTURE - HIGH LEVEL (APPLE) EVENT

Event ID

The event ID identifies the particular event within the event class, uniquely identifying the Apple event and communicating the action the Apple event should perform. As shown at Fig 2, the event ID appears in the `where` field of the event structure for an Apple event. For example, for an Open Documents event, the where field will contain the value `'odoc'` (which is represented by the constant `kAEOpenDocuments`.)

The following are the event IDs for the five required Apple events and the four Appearance Manager Apple events.

Event ID	Value	Description
`kAEOpenApplication`	`'oapp'`	Perform those tasks associated with the user opening the application.
`kAEReopenApplication`	`'rapp'`	Perform those tasks associated with the user "re-opening" the application.
`kAEOpenDocuments`	`'odoc'`	Open documents.
`kAEPrintDocuments`	`'pdoc'`	Print Documents.
`kAEQuitApplication`	`'quit'`	Quit the application.
`kAEAppearanceChanged`	`'thme'`	Current appearance has changed. Action as required.
`kAESystemFontChanged`	`'sysf'`	Current system font has changed. Action as required.
`kAESmallSystemFontChanged`	`'ssfn'`	Current small system font has changed. Action as required.
`kAEViewsFontChanged`	`'vfnt'`	Current views font has changed. Action as required.
`kAEShowPreferences`	`'pref'`	User has chosen the Prefences item in the Mac OS X Application menu.

Target Application

In addition to the event class and event ID, every Apple event must include an attribute which specifies the target application's address.

Apple Event Parameters

An Apple event parameter is a structure containing data used by the target application. Apple events can use standard data types, such as strings of text, long integers, boolean values, and alias structures, for the data in their parameters.

There are various kinds of Apple event parameters, including **direct parameters** and **additional parameters**.

Direct Parameters

Direct parameters usually specify the data to be acted upon by the target application. For example, a list of documents is contained in the direct parameter of the Open Documents event.

Additional Parameters

Some Apple events also take additional parameters. The target application uses these additional parameters (for example, operands in an equation) in addition to the data specified in the direct parameter.

Required and Optional Parameters

All parameters are described as either **required parameters** or **optional parameters** in the Apple Event Registry: Standard Suites. Direct parameters are usually defined as required parameters.

Attributes and Parameters in an Open Documents Apple Event

Fig 3 shows the major Apple event attributes and the direct parameter for the Open Documents event.

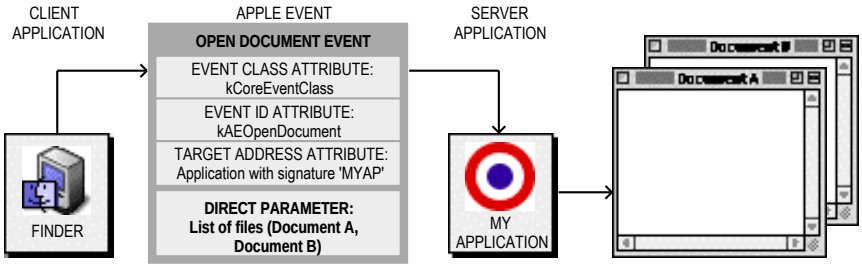

FIG 3 - OPEN DOCUMENTS APPLE EVENT - MAJOR ATTRIBUTES AND DIRECT PARAMETER

To process this event, the application My Application would use the `AEProcessAppleEvent` function, which uses the event class and event ID attributes to dispatch the event to its Open Documents handler. The Open Documents handler opens the documents specified in the direct parameter.

Data Structures Within Apple Events

The Apple Event Manager constructs its own internal data structures to contain the information in an Apple event.

Descriptor Structures

The Apple Event Manager uses **descriptor structures**, which you can think of as building blocks, to construct Apple event attributes and parameters. Descriptor structures comprise a handle to data and a **descriptor type**. The descriptor type identifies the type of data to which the handle refers:

```
struct AEDesc
{
  DescType      descriptorType;  // Type of data.
  AEDataStorage dataHandle;      // Handle to data.
};
typedef struct AEDesc AEDesc;
```

> ***Carbon Note***
>
> In Carbon, the `dataHandle` field is opaque. You must use the accessor functions `AEGetDescData` and `AEGetDescDataSize` to access the data in this field.

The descriptor type is of type `DescType`, that is, a four-character code. The following are the codes for some of the major descriptor types, the constants used to represent them, and the kind of data they identify:

Descriptor Type	Value	Description of Data
`typeChar`	`'TEXT'`	Unterminated string.
`typeType`	`'type'`	Four-character code.
`typeBoolean`	`'bool'`	One-byte Boolean value.
`typeLongInteger`	`'long'`	32-bit integer.
`typeAEList`	`'list'`	List of descriptor structures.
`typeAERecord`	`'reco'`	List of keyword-specified descriptor structures.
`typeAppleEvent`	`'aevt'`	Apple event structure.
`typeFSS`	`'fss '`	File system specification.
`typeKeyword`	`'keyw'`	Apple event keyword.
`typeNull`	`'null'`	Nonexistent data (handle whose value is `NULL`).

Fig 4 illustrates the logical arrangement of two descriptor structures. The first specifies that the data handle refers to an unterminated string. The second specifies that the data handle refers to a four-character code, in this case `'aevt'` (which is represented by the constant `kCoreEventClass`).

FIG 4 - TYPICAL DESCRIPTOR STRUCTURES

Address Descriptor Structures

Every Apple event includes an attribute specifying the address of the target application. A descriptor structure which contains an application's address is called an **address descriptor structure**:

```
typedef AEDesc AEAddressDesc;  // An AEDesc which contains addressing data.
```

Keyword-Specified Descriptor Structures

The Apple Event Manager assembles the various descriptor structures into an Apple event. Your application cannot access the contents of the Apple event directly; rather, Event Manager functions must be used to request each attribute and parameter by **keyword**. Keywords, which are four-character codes of type `AEKeyword`, are used to keep track of various descriptor structures.

The following are typical keywords and the constants used to represent them:

Keyword	Value	Description
keyMissedKeywordAttr	'miss'	For first required parameter remaining in an Apple event.
keyDirectObject	'----'	Direct parameter.

Keywords are associated with specific descriptor structures by means of **keyword-specified descriptor structures**:

```
struct AEKeyDesc
{
  AEKeyword descKey;      // Keyword.
  AEDesc    descContent;  // Descriptor structure.
};
typedef struct AEKeyDesc AEKeyDesc;
```

Descriptor Lists, AE Structures, and AppleEvent Structures

Descriptor Lists

To extract data from an Apple event, you use Apple Event Manager functions to either copy data to a buffer, return a descriptor structure whose data handle refers to a copy of the data, or return **descriptor lists**, which are lists of descriptor structures. A descriptor list is a descriptor structure whose handle refers to a list of other descriptor structures (unless it is an empty list):

```
typedef AEDesc AEDescList;  // List of descriptor structures.
```

Fig 5 illustrates the logical arrangement of the descriptor list that specifies the direct parameter of the Open Documents event shown at Fig 3.

Data Type AEDescList

Descriptor type:	`typeAEList`
Data:	List of descriptor records

Descriptor type:	`typeAlias`
Data:	Alias record for filename (Document A)

Descriptor type:	`typeAlias`
Data:	Alias record for filename (Document B)

FIG 5 - A DESCRIPTOR LIST FOR A LIST OF ALIASES

This descriptor list provides the data for a keyword-specified descriptor structure.

AE Structure

Keyword-specified descriptor structures can in turn be combined into an **AE structure**. An AE structure is a descriptor list of type `AERecord`:

```
typedef AEDescList AERecord;  // List of keyword-specified descriptor structures.
```

The handle for a descriptor list of type `AERecord` refers to a list of keyword-specified descriptor structures that can be used to construct Apple event parameters.

Apple Event Structure

An **Apple event structure** is another special descriptor list of type `AppleEvent`:

```
typedef AERecord AppleEvent;  // List of attributes and parameters for Apple event.
```

An Apple event structure describes an Apple event. The data for an Apple event structure, like the data for an AE structure, consists of a list of keyword-specified descriptor structures. The difference between an AE structure and an Apple event structure is that the data in the latter is divided into two parts, the first for attributes and the second for parameters.

Passing Descriptor Lists, AE Structures and Apple Event Structures to Apple Event Manager Functions

You can pass an Apple event structure to any Apple Event Manager function that expects an AE structure, and you can pass Apple event structures and AE structures, as well as descriptor lists and descriptor structures, to any Apple Event Manager functions that expect structures of data type `AEDesc`.

Example Complete Apple Event

Fig 6 shows an example of a complete Apple event. This is a data structure of type `AppleEvent` which contains a list of keyword-specified descriptor structures containing the attributes and parameters of an Open Documents event.

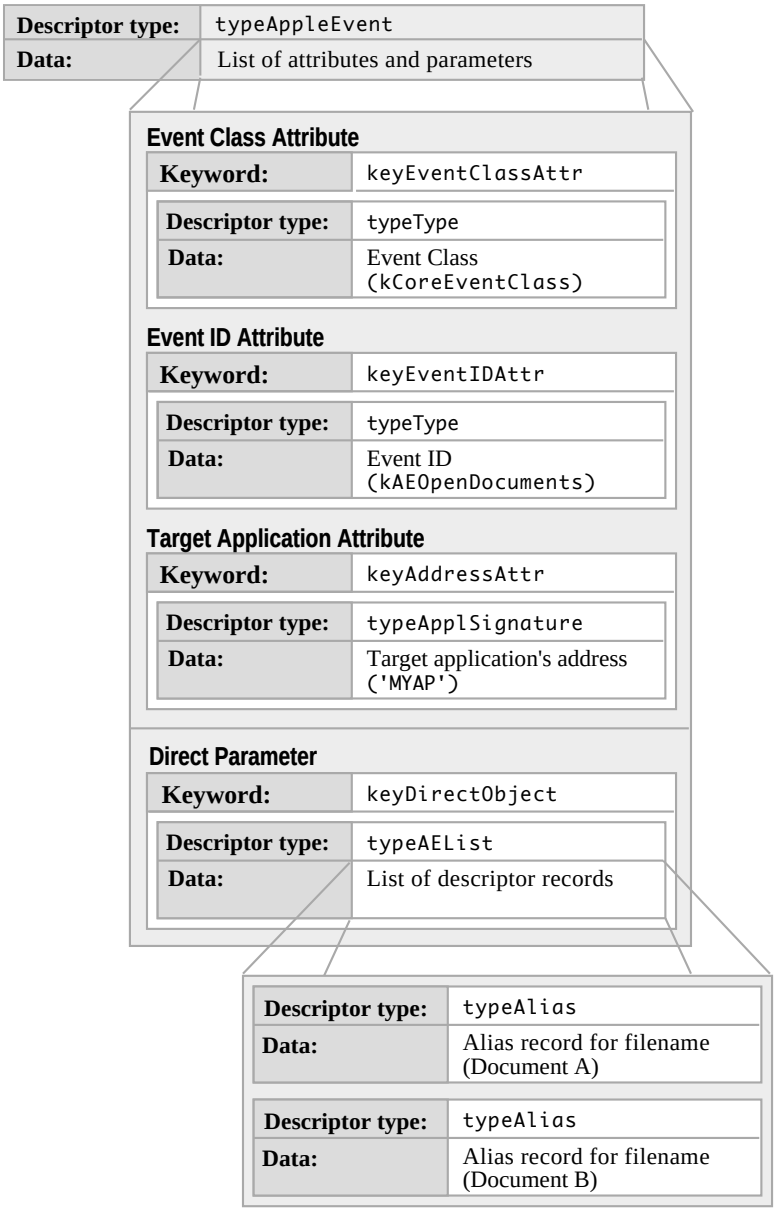

FIG 6 - AN APPLE (OPEN DOCUMENTS) EVENT

Handling Apple Events

To request a service or information, a client application uses the Apple Event Manager to create and send an Apple event. To respond, a server application uses the Apple Event Manager to extract data from the attributes and parameters of the Apple event. In addition, and where necessary, the server application adds requested data to the reply event returned to the client by the Apple Event Manager.

As previously stated, your application should, at the very least, support the required Apple events sent by the Finder. Your application must therefore:

- Set the `isHighLevelEventAware` flag in your application's `'SIZE'` resource.
- Test for high-level events in the event loop. An Apple event (like all high-level events) is identified by a message class of `kHighLevelEvent` in the `what` field of the event structure.

- Use `AEProcessAppleEvent` to process the Apple events. `AEProcessAppleEvent` examines the data in the event class and event ID attributes so as to identify the Apple event and then calls the appropriate **Apple event handler** provided by your application.
- Provide handlers for the required Apple events in your application. Your Apple event handlers must extract the pertinent data from the Apple event, perform the requested action, and return a result.
- Use `AEInstallEventHandler` to install your Apple event handlers in an **Apple event dispatch table** for your application. The Apple event dispatch table is used by the Apple Event Manager to map Apple events to your application's handlers. Calls to `AEProcessAppleEvent` cause the Apple Event Manager to check the dispatch table and, if your application has installed a handler for the event, call the handler.

Apple Event Handlers

Each Apple event handler must be a function which uses this syntax:

```
OSErr theEventHandler(AppleEvent *appleEvent,AppleEvent *reply,long handlerRefcon);
```

`appleEvent`	The Apple event to handle. Your handler uses Apple Event Manager functions to extract any parameters and attributes from the Apple event and then perform the necessary processing.
`reply`	The default reply provided by the Apple Event Manager.
`handlerRefcon`	Reference constant stored in the Apple event dispatch table entry for the Apple event. Your handler can ignore this parameter if your application does not use the reference constant.

Apple event handlers must generally perform the following tasks:

- Extract the attributes and parameters from the Apple event.
- Check that all required parameters have been extracted.
- Perform the action requested by the Apple event.
- Dispose of any copies of the descriptor structures that have been created.
- Add information to the reply Apple event if requested.

Extracting and Checking Data

You must use Apple Event Manager functions to extract the data from Apple events. The following are the main functions involved:

Function	Description
`AEGetAttributePtr`	Uses a buffer to return a copy of the data contained in an Apple event attribute. Used to extract data of fixed length or known maximum length.
`AEGetParamDesc`	Returns a copy of the descriptor structure or descriptor list for an Apple event parameter. Usually used to extract data of variable length, for example, to extract the descriptor list for a list of alias structures specified in the direct parameter of an Open Documents event.
`AECountItems`	Returns the number of descriptor structures in a descriptor list. Used, for example, to determine the number of alias structures for documents specified in the direct parameter of an Open Documents event.
`AEGetNthPtr`	Uses a buffer to return a copy of the data for a descriptor structure contained in a descriptor list. Used to extract data of fixed length or known maximum length, for example, to extract the name and location of a document from the descriptor list specified in the direct parameter of the Open Documents event.

Data Type Coercion

You can specify the descriptor type in the resulting data from these functions. If this type is different from the descriptor type of the attribute or parameter, the Apple Event Manager attempts to coerce it to the specified type. In the direct parameter of the Open Documents event, for example, each descriptor

structure in the descriptor list is an **alias structure** and each alias structure specifies a document to be opened. All your application usually needs to open a document is a **file system specification structure** (`FSSpec`) of the document. When you extract the descriptor structure from the descriptor list, you can request that the Apple Event Manager return the data to your application as a file system specification structure instead of an alias structure.

Checking That All Required Parameters Have Been Retrieved

After extracting all known Apple event parameters, your handler should check that it has retrieved all the parameters that the source application considered to be required. To do this, determine whether the `keyMissedKeywordAttr` attribute exists. If this attribute does exist, your handler has not retrieved all the required parameters, and it should return an error.

Performing the Requested Action and Returning a Result

When your application responds to an Apple event, it should perform the standard action requested by the event.

Your Apple event handler should always set its function result to either `noErr`, if it successfully handles the Apple event, or to a non-zero result code if an error occurs. When your handler returns a non-zero result code, the Apple Event Manager adds a `keyErrorNumber` parameter, containing the result code that your handler returns, to the reply Apple event.

Disposing of Copies of Descriptor Structures

When your handler is finished with a copy of a descriptor structure created by `AEGetParamDesc` and related functions, it should dispose of it by calling `AEDisposeDesc`.

Required Apple Events - Contents and Required Action

Your application receives the five required Apple events from the Finder in these circumstances:

- If your application is not open and the user elects to open it from the Finder without opening or printing any documents (either by double clicking the application's icon, selecting the icon and choosing **Open** from the Finder's **File** menu, or choosing it from the Mac OS 8/9 Apple menu), the Finder calls the Process Manager to launch your application and then sends your application an Open Application event.
- If your application is already open and the user attempts to "open" it again, the Finder sends your application a Re-open Application event.[1]
- If your application is not open and the user elects to open one of your application's documents from the Finder, the Finder calls the Process Manager to launch your application and then sends your application an Open Documents event.
- On Mac OS 8/9, if your application is not open and the user elects to print one of your application's documents from the Finder, the Finder calls the Process Manager to launch your application and then sends your application a Print Documents event. Your application should print the selected documents and remain open until it receives a Quit Application event from the Finder.
- If your application is open and the user elects to open or (on Mac OS 8/9 only) print any of your application's documents from the Finder, the Finder sends your application the Open Documents or (on Mac OS 8/9 only) Print Documents event.

[1] The Re-open Application event was introduced with MAC OS 8 to cater for a situation which could confuse inexperienced users. The specific situation is where the application is open but has no open windows. Because of the absence of a window, the user does not realise that the application is running, attempts to "open" it from the Finder, and then fails to notice the menu bar change. The intention of the Re-open Application event in such circumstances is to cause the application to open a window, providing more obvious visible evidence to the user that the application is, in fact, open.

- If your application is open and the user chooses **Restart** or **Shut Down**, the Finder sends your application the Quit Application event.

The following is a summary of the contents of the required Apple events sent by the Finder and the actions they request applications to perform:

Open Application event	
Attributes:	
Event Class:	`kCoreEventClass`
Event ID:	`kAEOpenApplication`
Parameters:	None.
Requested Action:	Perform tasks your application normally performs when a user opens your application without opening or printing any documents, such as opening an untitled document window.
Re-open Application event	
Attributes:	
Event Class:	`kCoreEventClass`
Event ID:	`kAEReopenApplication`
Parameters:	None.
Requested Action:	If no windows are currently open, open a new untitled document window.
Open Documents event	
Attributes:	
Event Class:	`kCoreEventClass`
Event ID:	`kAEOpenDocuments`
Required parameters:	
Keyword:	`keyDirectObject`
Descriptor type:	`typeAEList`
Data:	A list of alias structures for the documents to be opened.
Requested Action:	Open the documents specified in the `keyDirectObject` parameter.
Print Documents event	
Attributes:	
Event Class:	`kCoreEventClass`
Event ID:	`kAEPrintDocuments`
Required parameters:	
Keyword:	`keyDirectObject`
Descriptor type:	`typeAEList`
Data:	A list of alias structures for the documents to be printed.
Requested action:	Print the documents specified in the `keyDirectObject` parameter.
Quit Application event	
Attributes:	
Event Class:	`kCoreEventClass`
Event ID:	`kAEQuitApplication`
Parameters:	None
Requested Action:	Perform any tasks that your application would normally perform when the user chooses **Quit** from the application's **File** menu. (Such tasks typically include releasing memory and asking the user whether to save documents which have been changed.)

Your application needs to recognise two descriptor types to handle the required Apple events: descriptor lists and alias structures.

As previously stated, in the event of an Open Documents or (on Mac OS 8/9 only) Print Documents event, you can retrieve the data which specifies the document as an alias structure, or you can request that the Apple Event Manager coerce the alias structure to a file system specification structure. The file system specification provides a standard method of identifying files.

Main Apple Event Manager and Appearance Manager Constants, Data Types, and Functions Relevant to Required Apple Events and Appearance Manager Apple Events

Constants

High Level Event

```
kHighLevelEvent           = 23
```

Event Classes for Required Apple Event and Appearance Manager Apple Event

```
kCoreEventClass           = FOUR_CHAR_CODE('aevt')  Event class - required Apple events.
KAppearanceEventClass     = FOUR_CHAR_CODE('appr')  Event Class - Appearance Manager Apple
                                                    events.
```

Event IDs for Required Apple Events

```
kAEOpenApplication        = FOUR_CHAR_CODE('oapp')  Event ID for Open Application event.
kAEReopenApplication      = FOUR_CHAR_CODE('rapp')  Event ID for Re-open Application Event.
kAEOpenDocuments          = FOUR_CHAR_CODE('odoc')  Event ID for Open Documents event.
kAEPrintDocuments         = FOUR_CHAR_CODE('pdoc')  Event ID for Print Documents event.
kAEQuitApplication        = FOUR_CHAR_CODE('quit')  Event ID for Quit Application event.
```

Event IDs for Appearance Manager Apple Events

```
kAESystemFontChanged      = FOUR_CHAR_CODE('sysf')  System font changed.
KAESmallSystemFontChanged = FOUR_CHAR_CODE('ssfn')  Small system font changed.
kAEViewsFontChanged       = FOUR_CHAR_CODE('vfnt')  Views font changed.
```

Event ID for Show Preferences Apple Event (Mac OS X)

```
kAEShowPreferences        = FOUR_CHAR_CODE('pref')  Preferences menu item chosen
```

Keywords for Apple Event Attributes

```
keyMissedKeywordAttr      = FOUR_CHAR_CODE('miss')  First required parameter remaining in
                                                    an Apple event.
```

Keywords for Apple Event Parameters

```
keyDirectObject           = FOUR_CHAR_CODE('----')  Direct parameter
```

Apple Event Descriptor Types

```
typeAEList                = FOUR_CHAR_CODE('list')  List of descriptor structures.
typeWildCard              = FOUR_CHAR_CODE('****')  Matches any type.
typeFSS                   = FOUR_CHAR_CODE('fss ')  File system specification.
```

Result Codes

```
errAEDescNotFound         = -1701  Descriptor structure was not found.
errAEParamMissed          = -1715  Handler cannot understand a parameter
                                   the client considers is required.
```

Theme Font ID Constants

```
KThemeSystemFont                = 0
KThemeSmallSystemFont           = 1
KThemeSmallEmphasizedSystemFont = 2
KThemeViewsFont                 = 3
```

Data Types

```
typedef FourCharCode  AEEventClass;  // Event class for a high level event.
typedef FourCharCode  AEEventID;     // Event ID for a high level event.
typedef FourCharCode  AEKeyword;     // Keyword for a descriptor structure.
typedef ResType       DescType;      // Descriptor type.
typedef AEDesc        AEDescList;    // List of descriptor structures.
typedef AEDescList    AERecord;      // List af keyword-specified descriptor structures.
typedef AERecord      AppleEvent;    // List of attributes and parameters for Apple event.
```

Descriptor Structure

```
struct AEDesc
{
  DescType      descriptorType;  // Type of data being passed.
  AEDataStorage dataHandle;      // Handle to data being passed.
};
typedef struct AEDesc AEDesc;
```

Keyword-Specified Descriptor Structure

```
struct AEKeyDesc
{
  AEKeyword deskKey;        // Keyword.
  AEDesc    descContent;    // Descriptor structure.
};
typedef struct AEKeyDesc AEKeyDesc;
```

Functions

Creating and Managing Apple Event Dispatch Tables

```
OSErr  AEInstallEventHandler(AEEventClass theAEEventClass,AEEventID theAEEventID,
       AEEventHandlerUPP handler,long handlerRefcon,Boolean isSysHandler);
```

Dispatching Apple Events

```
OSErr  AEProcessAppleEvent(const EventRecord *theEventRecord);
```

Getting Data or Descriptor Structures Out of Apple Event Parameters and Attributes

```
OSErr  AEGetParamDesc(const AppleEvent *theAppleEvent,AEKeyword theAEKeyword,
       DescType desiredType,AEDesc *result);
OSErr  AEGetAttributePtr(const AppleEvent *theAppleEvent,AEKeyword theAEKeyword,
       DescType desiredType,DescType *typeCode,Ptr dataPtr,Size maximumSize,
       Size *actualSize);
```

Counting the Items in Descriptor Lists

```
OSErr  AECountItems(const AEDescList *theAEDescList,long *theCount);
```

Getting Items From Descriptor Lists

```
OSErr  AEGetNthPtr(const AEDescList *theAEDescList,long index,DescType desiredType,AEKeyword
       *theAEKeyword,DescType *typeCode,Ptr dataPtr,Size maximumSize,Size *actualSize);
```

Deallocating Memory for Descriptor Structures

```
OSErr  AEDisposeDesc(AEDesc *theAEDesc);
```

```
// ****************************************************************************************
// AppleEvents.c                                                        CLASSIC EVENT MODEL
// ****************************************************************************************
//
// This program:
//
// •  Installs handlers for the required Apple events, Appearance Manager Apple events, and,
//    on Mac OS X only, the Show Preferences Apple event.
//
// •  Responds to the receipt of required Apple events by displaying descriptive text in a
//    window opened for that purpose, and by opening simulated document windows as
//    appropriate.  These responses result from the user:
//
//    •   Double clicking on the application's icon, or selecting the icon and choosing Open
//        from the Finder's File menu, thus causing the receipt of an Open Application event.
//
//    •   When the application is already open, double clicking on the application's icon, or
//        selecting the icon and choosing Open from the Finder's File menu, thus causing the
//        receipt of a Re-open  Application event.
//
//    •   Double clicking on one of the document icons, selecting one or both of the document
//        icons and choosing Open from the Finder's File menu, or dragging one or both of the
//        document icons onto the application's icon, thus causing the receipt of an Open
//        Documents event.
//
//    •   On Mac OS 8/9, selecting one or both of the document icons and choosing Print from
//        the Finder's file menu, thus causing the receipt of a Print Documents event and, if
//        the application is not already running, a subsequent Quit Application event.
//
//    •   While the application is running, choosing Shut Down or Restart from the Finder's
//        Special menu, thus causing the receipt of a Quit Application event.
//
// •  Responds to the receipt of Appearance Manager Apple events (Mac OS 8/9) and the Show
//    Preferences Apple event (Mac OS X) by displaying descriptive text.
//
// The program, which is intended to be run as a built application rather than within
// CodeWarrior, utilises the following resources:
//
// •  A 'plst' resource containing an information property list which provides information
//    to the Mac OS X Finder.
//
// •  An 'icns' resource containing application and document icons for Mac OS X.
//
// •  'WIND' resources (purgeable, initially visible) for the descriptive text display window
//     and simulated document windows.
//
// •  'MBAR' and 'MENU' resources (preload, non-purgeable).
//
// •  'STR#' resources (purgeable) for displaying error messages using StandardAlert.
//
// •  For Mac OS 8/9:
//
//    •   'ICN#', 'ics#', 'ics4', 'ics8', 'icl4', and 'icl8' resources (that is, an icon
//        family) for the application and for the application's documents. (Purgeable.)
//
//    •   'FREF' resources (non-purgeable) for the application and the application's 'TEXT'
//        documents, which link the icons with the file types they represent, and which allow
//        users to launch the application by dragging the document icons to the application
//        icon.
//
//    •   The application's signature resource (non-purgeable), which enables the Finder to
//        identify and start up the application when the user double clicks the application's
//        document icons.
//
//    •   A 'BNDL' resource (non-purgeable), which groups together the application's
//        signature, icon and 'FREF' resources.
```

```
//
//    •   A 'hfdr' resource (purgeable), which provides the customised finder icon help
//        override help balloon for the  application icon.
//
//    •   A 'vers' resource (purgeable), which provides version information via the Show Info
//        window and the Version column in list view windows.
//
// •  A 'SIZE' resource with the acceptSuspendResumeEvents, canBackground,
//    doesActivateOnFGSwitch, and isHighLevelEventAware flags set.
//
// ********************************************************************************************

// ..... includes

#include <Carbon.h>

// ..... defines

#define rMenubar             128
#define mFile                129
#define  iQuit               12
#define rDisplayWindow       128
#define rDocWindow           129
#define rErrorStrings        128
#define eInstallHandler      1
#define eGetRequiredParam    2
#define eGetDescriptorRecord 3
#define eMissedRequiredParam 4
#define eCountDescripRecords 5
#define eCannotOpenFile      6
#define eCannotPrintFile     7
#define eCannotOpenWindow    8
#define eMenus               9
#define MIN(a,b)             ((a) < (b) ? (a) : (b))

// ..... global variables

WindowRef gWindowRef;
Boolean   gRunningOnX = false;
Boolean   gDone;
Boolean   gApplicationWasOpen = false;

// ..... function prototypes

void      main                           (void);
void      doPreliminaries                (void);
void      doInstallAEHandlers            (void);
void      doInstallAnAEHandler           (AEEventClass,AEEventID,void *);
void      doEvents                       (EventRecord *);
OSErr     openAppEventHandler            (AppleEvent *,AppleEvent *,SInt32);
OSErr     reopenAppEventHandler          (AppleEvent *,AppleEvent *,SInt32);
OSErr     openDocsEventHandler           (AppleEvent *,AppleEvent *,SInt32);
OSErr     printDocsEventHandler          (AppleEvent *,AppleEvent *,SInt32);
OSErr     quitAppEventHandler            (AppleEvent *,AppleEvent *,SInt32);
OSErr     sysFontChangeEventHandler      (AppleEvent *,AppleEvent *,SInt32);
OSErr     smallSysFontChangeEventHandler (AppleEvent *,AppleEvent *,SInt32);
OSErr     viewsFontChangeEventHandler    (AppleEvent *,AppleEvent *,SInt32);
OSErr     showPreferencesEventHandler    (AppleEvent *,AppleEvent *,SInt32);
OSErr     doHasGotRequiredParams         (AppleEvent *);
Boolean   doOpenFile                     (FSSpec *,SInt32,SInt32);
Boolean   doPrintFile                    (FSSpec *,SInt32,SInt32);
void      doPrepareToTerminate           (void);
WindowRef doNewWindow                    (void);
void      doMenuChoice                   (SInt32);
void      doErrorAlert                   (SInt16);
void      doDrawText                     (Str255);
void      doConcatPStrings               (Str255,Str255);

// ********************************************************************************** main
```

```
void  main(void)
{
  Rect          portRect;
  RGBColor      foreColour = { 0xFFFF,0xFFFF,0xFFFF };
  RGBColor      backColour = { 0x4444,0x4444,0x9999 };
  MenuBarHandle menubarHdl;
  SInt32        response;
  MenuRef       menuRef;
  EventRecord   eventStructure;

  // ..... do preliminaries

  doPreliminaries();

  // ..... set up menu bar and menus

  menubarHdl = GetNewMBar(rMenubar);
  if(menubarHdl == NULL)
    doErrorAlert(eMenus);
  SetMenuBar(menubarHdl);
  DrawMenuBar();

  Gestalt(gestaltMenuMgrAttr,&response);
  if(response & gestaltMenuMgrAquaLayoutMask)
  {
    menuRef = GetMenuRef(mFile);
    if(menuRef != NULL)
    {
      DeleteMenuItem(menuRef,iQuit);
      DeleteMenuItem(menuRef,iQuit - 1);
      DisableMenuItem(menuRef,0);
    }

    gRunningOnX = true;

    EnableMenuCommand(NULL,kAEShowPreferences);
  }

  // ..... open a window

  if(!(gWindowRef = GetNewCWindow(rDisplayWindow,NULL,(WindowRef) -1)))
  {
    doErrorAlert(eCannotOpenWindow);
    ExitToShell();
  }

  SetPortWindowPort(gWindowRef);
  TextSize(10);
  TextFace(bold);
  RGBBackColor(&backColour);
  RGBForeColor(&foreColour);
  GetWindowPortBounds(gWindowRef,&portRect);
  EraseRect(&portRect);

  // ..... install Apple event handlers

  doInstallAEHandlers();

  // ..... event loop

  gDone = false;

  while(!gDone)
  {
    if(WaitNextEvent(everyEvent,&eventStructure,180,NULL))
      doEvents(&eventStructure);
  }
}
```

```
// ********************************************************************** doPreliminaries

void  doPreliminaries(void)
{
  MoreMasterPointers(64);
  InitCursor();
  FlushEvents(everyEvent,0);
}

// ********************************************************************** doInstallAEHandlers

void  doInstallAEHandlers(void)
{
  // ..... required Apple events

  doInstallAnAEHandler(kCoreEventClass,kAEOpenApplication,openAppEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEReopenApplication,reopenAppEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEOpenDocuments,openDocsEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEPrintDocuments,printDocsEventHandler);
  doInstallAnAEHandler(kCoreEventClass,kAEQuitApplication,quitAppEventHandler);

  // ..... Appearance Manager Apple events

  doInstallAnAEHandler(kAppearanceEventClass,kAESystemFontChanged,sysFontChangeEventHandler);
  doInstallAnAEHandler(kAppearanceEventClass,kAESmallSystemFontChanged,
                       smallSysFontChangeEventHandler);
  doInstallAnAEHandler(kAppearanceEventClass,kAEViewsFontChanged,viewsFontChangeEventHandler);

  // ..... Show Preferences Apple event

  if(gRunningOnX)
    doInstallAnAEHandler(kCoreEventClass,kAEShowPreferences,showPreferencesEventHandler);
}

// ********************************************************************** doInstallAnAEHandler

void  doInstallAnAEHandler(AEEventClass eventClass,AEEventID eventID,void *theHandler)
{
  OSErr  osError;

  osError = AEInstallEventHandler(eventClass,eventID,
                                  NewAEEventHandlerUPP((AEEventHandlerProcPtr) theHandler),
                                  0L,false);
  if(osError != noErr)
    doErrorAlert(eInstallHandler);
}

// ********************************************************************************** doEvents

void  doEvents(EventRecord *eventStrucPtr)
{
  WindowPartCode partCode;
  WindowRef      windowRef;
  SInt32         menuChoice;

  switch(eventStrucPtr->what)
  {
    case kHighLevelEvent:
      AEProcessAppleEvent(eventStrucPtr);
      break;

    case mouseDown:
      partCode = FindWindow(eventStrucPtr->where,&windowRef);
      switch(partCode)
      {
        case inMenuBar:
          menuChoice = MenuSelect(eventStrucPtr->where);
          doMenuChoice(menuChoice);
```

```
          break;

        case inDrag:
          DragWindow(windowRef,eventStrucPtr->where,NULL);
          break;

        case inContent:
          if(windowRef != FrontWindow())
            SelectWindow(windowRef);
          break;

        case inGoAway:
          DisposeWindow(windowRef);
          break;
      }
      break;

    case keyDown:
      if((eventStrucPtr->modifiers & cmdKey) != 0)
        doMenuChoice(MenuEvent(eventStrucPtr));
      break;

    case updateEvt:
      BeginUpdate((WindowRef)eventStrucPtr->message);
      EndUpdate((WindowRef)eventStrucPtr->message);
      break;
  }
}

// ********************************************************************** openAppEventHandler

OSErr  openAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr     osError;
  WindowRef windowRef;

  gApplicationWasOpen = true;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    doDrawText("\pReceived an Apple event: OPEN APPLICATION.");
    doDrawText("\p    • Opening an untitled window in response.");

    windowRef = doNewWindow();
    SetWTitle(windowRef,"\puntitled");
  }

  return osError;
}

// ********************************************************************** reopenAppEventHandler

OSErr  reopenAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefCon)
{
  OSErr     osError;
  WindowRef windowRef;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    doDrawText("\pReceived an Apple event: RE-OPEN APPLICATION.");
    doDrawText("\p    • I will check whether I have any windows open.");

    windowRef = GetWindowList();

    if((windowRef = GetNextWindow(windowRef)) == NULL)
```

```
    {
      doDrawText("\p      No windows are open, so I will open a window.");

      windowRef = doNewWindow();
      SetWTitle(windowRef,"\puntitled 1");
    }
    else
      doDrawText("\p      A window is open, so I won't open another.");
  }

  return osError;
}

// ********************************************************************** openDocsEventHandler

OSErr  openDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  AEDescList docList;
  OSErr      osError, ignoreErr;
  SInt32     numberOfItems, index;
  DescType   returnedType;
  FSSpec     fileSpec;
  AEKeyword  keyWord;
  Size       actualSize;
  Boolean    result;

  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,&fileSpec,
                                sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            if(!(result = doOpenFile(&fileSpec,index,numberOfItems)))
              doErrorAlert(eCannotOpenFile);
          }
          else
            doErrorAlert(eGetDescriptorRecord);
        }
      }
      else
        doErrorAlert(eCountDescripRecords);
    }
    else
      doErrorAlert(eMissedRequiredParam);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(eGetRequiredParam);

  return osError;
}

// ********************************************************************** printDocsEventHandler

OSErr  printDocsEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  AEDescList docList;
  OSErr      osError, ignoreErr;
```

```
  SInt32     numberOfItems, index;
  DescType   returnedType;
  FSSpec     fileSpec;
  AEKeyword  keyWord;
  Size       actualSize;
  Boolean    result;

  osError = AEGetParamDesc(appEvent,keyDirectObject,typeAEList,&docList);

  if(osError == noErr)
  {
    osError = doHasGotRequiredParams(appEvent);
    if(osError == noErr)
    {
      osError = AECountItems(&docList,&numberOfItems);
      if(osError == noErr)
      {
        for(index=1;index<=numberOfItems;index++)
        {
          osError = AEGetNthPtr(&docList,index,typeFSS,&keyWord,&returnedType,&fileSpec,
                                sizeof(fileSpec),&actualSize);
          if(osError == noErr)
          {
            if(!(result = doPrintFile(&fileSpec,index,numberOfItems)))
              doErrorAlert(eCannotPrintFile);
          }
          else
            doErrorAlert(eGetDescriptorRecord);
        }
      }
      else
        doErrorAlert(eCountDescripRecords);
    }
    else
      doErrorAlert(eMissedRequiredParam);

    ignoreErr = AEDisposeDesc(&docList);
  }
  else
    doErrorAlert(eGetRequiredParam);

  return osError;
}

// ********************************************************************** quitAppEventHandler

OSErr  quitAppEventHandler(AppleEvent *appEvent,AppleEvent *reply,SInt32 handlerRefcon)
{
  OSErr osError;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
    doPrepareToTerminate();

  return osError;
}

// ************************************************************** sysFontChangeEventHandler

OSErr  sysFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                 SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, theString = "\p   Current large system font is: ";

  osError = doHasGotRequiredParams(appEvent);
```

```
  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: LARGE SYSTEM FONT CHANGED.");
    GetThemeFont(kThemeSystemFont,smSystemScript,fontName,NULL,NULL);
    doConcatPStrings(theString,fontName);
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}

// ********************************************************** smallSysFontChangeEventHandler

OSErr  smallSysFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                      SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, theString = "\p   Current small system font is: ";

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: SMALL SYSTEM FONT CHANGED.");
    GetThemeFont(kThemeSmallSystemFont,smSystemScript,fontName,NULL,NULL);
    doConcatPStrings(theString,fontName);
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}

// ************************************************************** viewsFontChangeEventHandler

OSErr  viewsFontChangeEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                   SInt32 handlerRefcon)
{
  OSErr  osError;
  Rect   portRect;
  Str255 fontName, fontSizeString, theString = "\p   Current views font is: ";
  SInt16 fontSize;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: VIEWS FONT CHANGED.");
    GetThemeFont(kThemeViewsFont,smSystemScript,fontName,&fontSize,NULL);
    doConcatPStrings(theString,fontName);
    doConcatPStrings(theString,"\p ");
    NumToString((SInt32) fontSize,fontSizeString);
    doConcatPStrings(theString,fontSizeString);
    doConcatPStrings(theString,"\p point");
    doDrawText(theString);
    // Action as required by application.
  }

  return osError;
}
```

```
// ************************************************************ showPreferencesEventHandler

OSErr  showPreferencesEventHandler(AppleEvent *appEvent,AppleEvent *reply,
                                   SInt32 handlerRefcon)
{
  OSErr osError;
  Rect  portRect;

  osError = doHasGotRequiredParams(appEvent);

  if(osError == noErr)
  {
    GetWindowPortBounds(gWindowRef,&portRect);
    EraseRect(&portRect);
    doDrawText("\pReceived an Apple event: SHOW PREFERENCES.");
    doDrawText("\p    • I would present a Preferences... dialog now.");
  }

  return osError;
}

// ********************************************************************* doHasGotRequiredParams

OSErr  doHasGotRequiredParams(AppleEvent *appEvent)
{
  OSErr    osError;
  DescType returnedType;
  Size     actualSize;

  osError = AEGetAttributePtr(appEvent,keyMissedKeywordAttr,typeWildCard,&returnedType,NULL,0,
                              &actualSize);

  if(osError == errAEDescNotFound)
    osError = noErr;
  else if(osError == noErr)
    osError = errAEParamMissed;

  return osError;
}

// ************************************************************************************ doOpenFile

Boolean  doOpenFile(FSSpec *fileSpecPtr,SInt32 index,SInt32 numberOfItems)
{
  WindowRef windowRef;

  gApplicationWasOpen = true;

  if(index == 1)
    doDrawText("\pReceived an Apple event: OPEN DOCUMENTS.");

  if(numberOfItems == 1)
  {
    doDrawText("\p    • The file to open is: ");
    DrawString(fileSpecPtr->name);
    doDrawText("\p    • Opening titled window in response.");
  }
  else
  {
    if(index == 1)
    {
      doDrawText("\p    • The files to open are: ");
      DrawString(fileSpecPtr->name);
    }
    else
    {
      DrawString("\p and ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p    • Opening titled windows in response.");
```

```
    }
  }

  if(windowRef = doNewWindow())
  {
    SetWTitle(windowRef,fileSpecPtr->name);
    return true;
  }
  else
    return false;
}

// ********************************************************************************** doPrintFile

Boolean  doPrintFile(FSSpec *fileSpecPtr,SInt32 index,SInt32 numberOfItems)
{
  WindowRef windowRef;
  UInt32    finalTicks;

  if(index == 1)
    doDrawText("\pReceived an Apple event: PRINT DOCUMENTS");

  if(numberOfItems == 1)
  {
    doDrawText("\p    • The file to print is: ");
    DrawString(fileSpecPtr->name);
    windowRef = doNewWindow();
    SetWTitle(windowRef,fileSpecPtr->name);
    Delay(60,&finalTicks);
    doDrawText("\p    • I would present the Print dialog first and then print");
    doDrawText("\p       the document when the user has made his settings.");
    Delay(60,&finalTicks);
    doDrawText("\p    • Assume that I am now printing the document.");
  }
  else
  {
    if(index == 1)
    {
      doDrawText("\p    • The first file to print is: ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p       I would present the Print dialog for the first file");
      doDrawText("\p       only and use the user's settings to print both files.");
    }
    else
    {
      doDrawText("\p    • The second file to print is: ");
      DrawString(fileSpecPtr->name);
      doDrawText("\p       I am using the Print dialog settings used for the");
      doDrawText("\p       first file.");
    }

    windowRef = doNewWindow();
    SetWTitle(windowRef,fileSpecPtr->name);
    doDrawText("\p    • Assume that I am now printing the document.");
  }

  if(numberOfItems == index)
  {
    if(!gApplicationWasOpen)
    {
      doDrawText("\p       Since the application was not already open, I expect to");
      doDrawText("\p       receive a QUIT APPLICATION event when I have finished.");
    }
    else
    {
      doDrawText("\p       Since the application was already open, I do NOT expect");
      doDrawText("\p       to receive a QUIT APPLICATION event when I have finished.");
    }
```

```
    Delay(180,&finalTicks);
    doDrawText("\p    • Finished print job.");
  }
  else
    Delay(180, &finalTicks);

  DisposeWindow(windowRef);
  return true;
}

// ********************************************************************** doPrepareToTerminate

void  doPrepareToTerminate(void)
{
  UInt32 finalTicks;

  doDrawText("\pReceived an Apple event: QUIT APPLICATION");

  if(gApplicationWasOpen)
  {
    doDrawText("\p    • I would now ask the user to save any unsaved files before");
    doDrawText("\p       terminating myself in response to the event.");
    doDrawText("\p    • Click the mouse when ready to terminate.");
    while(!Button()) ;
  }
  else
  {
    doDrawText("\p    • Terminating myself in response");
    Delay(240,&finalTicks);
  }

  // If the user did not click the Cancel button in a Save dialog:

  gDone = true;
}

// ********************************************************************************** doNewWindow

WindowRef  doNewWindow(void)
{
  WindowRef windowRef;

  if(!(windowRef = GetNewCWindow(rDocWindow,NULL,(WindowRef) -1)))
    doErrorAlert(eCannotOpenWindow);

  return windowRef;
}

// ********************************************************************************** doMenuChoice

void  doMenuChoice(SInt32 menuChoice)
{
  MenuID        menuID;
  MenuItemIndex menuItem;

  menuID   = HiWord(menuChoice);
  menuItem = LoWord(menuChoice);

  if(menuID == 0)
    return;

  switch(menuID)
  {
    case mFile:
      if(menuItem  == iQuit)
        gDone = true;
      break;
  }
```

```
  HiliteMenu(0);
}

// ********************************************************************************* doErrorAlert

void  doErrorAlert(SInt16 errorType)
{
  Str255 errorString;
  SInt16 itemHit;

  GetIndString(errorString,rErrorStrings,errorType);

  if(errorType < 7)
    StandardAlert(kAlertCautionAlert,errorString,NULL,NULL,&itemHit);
  else
  {
    StandardAlert(kAlertStopAlert,errorString,NULL,NULL,&itemHit);
    ExitToShell();
  }
}

// ********************************************************************************** doDrawText

void  doDrawText(Str255 eventString)
{
  RgnHandle tempRegion;
  SInt16    a;
  Rect      portRect;
  UInt32    finalTicks;

  tempRegion = NewRgn();
  GetWindowPortBounds(gWindowRef,&portRect);

  for(a=0;a<15;a++)
  {
    ScrollRect(&portRect,0,-1,tempRegion);
    QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);

    Delay(4,&finalTicks);
  }

  DisposeRgn(tempRegion);

  MoveTo(8,160);
  DrawString(eventString);
  QDFlushPortBuffer(GetWindowPort(gWindowRef),NULL);
}

// ******************************************************************************* doConcatPStrings

void  doConcatPStrings(Str255 targetString,Str255 appendString)
{
  SInt16 appendLength;

  appendLength = MIN(appendString[0],255 - targetString[0]);

  if(appendLength > 0)
  {
    BlockMoveData(appendString+1,targetString+targetString[0]+1,(SInt32) appendLength);
    targetString[0] += appendLength;
  }
}

// ************************************************************************************************
```

Demonstration Program AppleEvents Comments

The demonstration requires that the user open the windows containing the AppleEvents application icon and the two document file icons AppleEvents Document A and AppleEvents Document B.

Using all of the methods available in the Finder (that is, double clicking the icons, dragging document icons to the application icon, selecting the icons and choosing Open and (on Mac OS 8/9 only) Print from the Finder's File menu, choosing the application from the Apple menu), the user should launch the application, open the simulated documents and (on Mac OS 8/9 only) "print" the documents, noting the descriptive text printed in the non-document window in response to the receipt of the resulting Apple events.

When the application is running, the user should double-click the application icon, choose the application from the Mac OS 8/9 Apple menu, or click the application icon and choose Open from the Finder's File menu, noting the receipt of the Re-open Application event. This should be done when one or more "document" windows are open and when no "document windows are open.

The user should also choose Restart or Shut Down while the application is running, also noting the displayed text resulting from receipt of the Quit Application event. (On Mac OS X, the Quit Application event is also received when Quit is chosen from the Application menu.)

On Mac OS 8/9, opening and printing should be attempted (from the Finder's File menu) when the application is already running and when the application is not running.

On Mac OS X, the user should choose Show Preferences… from the Application menu.

Note that, in this demonstration, it is possible to "open" each "document" more than once, that is, it is possible to have several "Document A" and/or "Document B" windows open at once. (A real application, of course, would permit only one "Document A" and/or one "Document B" window to be open at any one time.) The reason for this is that, in this demonstration, files are not actually opened; accordingly, it is not possible to check whether the relevant file is already open or not.

With regard to the Appearance Manager Apple events, the user should make changes to the system font, small system font, and views font in the Fonts tab of the Appearance control panel, noting the descriptive text that appears in the non-document window.

Although not related to the required Apple events aspects of the program, the following aspects of the demonstration may also be investigated:

- On Mac OS 8/9, the customised finder icon help override help balloon for the application icon. (The 'hfdr' resource refers.)
- The version information for the application in the Finder's Get Info (Mac OS 8/9) and Show Info (Mac OS X) window and in the window containing the AppleEvents application when list view is selected.

'plst' Resource

The following is the information property list in the application's 'plst' resource:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist SYSTEM "file://localhost/System/Library/DTDs/PropertyList.dtd">
<plist version="0.9">
<dict>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>AppleEvents</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleSignature</key>
  <string>KJBB</string>
  <key>CFBundleIconFile</key>
  <string>128</string>
  <key>CFBundleIdentifier</key>
  <string>com.Windmill Software.AppleEvents</string>
  <key>LSPrefersCarbon</key>
  <string>Yes</string>
  <key>CFBundleLongVersionString</key>
  <string>Version 1.0.0 June 2001</string>
```

```
  <key>CFBundleShortVersionString</key>
  <string>1.0.0</string>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>NSAppleScriptEnabled</key>
  <string>No</string>
  <key>CFBundleDocumentTypes</key>
  <array>
    <dict>
      <key>CFBundleTypeIconFile</key>
      <string>129</string>
      <key>CFBundleTypeName</key>
      <string>AppleEvents Document</string>
      <key>CFBundleTypeOSTypes</key>
      <array>
        <string>TEXT</string>
      </array>
      <key>CFBundleTypeRole</key>
      <string>Viewer</string>
    </dict>
  </array>
</dict>
</plist>
```

Global Variables

gRunningOnX will be set to true if the program is running on Mac OS X.

gApplicationWasOpen will be used to control the manner of program termination when a Quit Application event is received, depending on whether the event followed a Print Documents event (on Mac OS 8/9) or resulted from the user choosing Restart of Shut Down from the Finder's Special menu.

main

In the menus setting up block, if the program is running on Mac OS X, the Preferences item in the Mac OS X Application menu is enabled.

The call to doInstallAEHandlers installs the Apple event handlers.

doInstallAEHandlers and doInstallAnAEHandler

doInstallAEHandlers, supported by doInstallAnAEHandler, installs the handlers for the required Apple events, the Appearance Manager Apple events, and (if the program is running on Mac OS X) the Show Preferences Apple event in the application's Apple event dispatch table.

In doInstallAnAEHandler, false is passed in AEInstallEventHandler's isSysHandler parameter. false causes the handler to be installed in the application's Apple event dispatch table. true causes handlers to be installed in the system's Apple event dispatch table. (The system Apple event dispatch table is a table in the system heap containing handlers that are available to all applications and processes running on the same computer.)

doEvents

The kHighLevelEvent case accommodates the receipt of a high-level event, in which case AEProcessAppleEvent is called. (AEProcessAppleEvent looks in the application's Apple event dispatch table for a match to the event class and event ID contained in, respectively, the event structure's message and where fields, and calls the appropriate handler.)

doOpenAppEvent

doOpenAppEvent is the handler for the Open Application event.

At the first line, the global variable gApplicationWasOpen, which controls the manner of program termination when a Quit Application event is received, is set to true. (This line is required for demonstration program purposes only.)

The function doHasGotRequiredParams is then called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, the Open Application event should not contain any required parameters.

If noErr is returned by doHasGotRequiredParams, the handler does what the user expects the application to do when it is opened, that is, it opens an untitled document window (the call to doNewWindow and the subsequent call to SetWTitle). The handler then returns noErr.

If errAEParamMissed is returned by doHasGotRequiredParams, this is returned by the handler.

The calls to doDrawText simply print some text in the text window for demonstration program purposes.

doReopenAppEvent

doRepenAppEvent is the handler for the Re-open Documents event.

At the first line, the function doHasGotRequiredParams is called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, the Re-open Application event should not contain any required parameters.

If noErr is returned by doHasGotRequiredParams, and if there are currently no open windows, the handler opens an untitled document window and returns noErr.

If errAEParamMissed is returned by doHasGotRequiredParams, this is returned by the handler.

The calls to doDrawText simply print some text in the text window for demonstration program purposes.

doOpenDocsEvent

doOpenDocsEvent is the handler for the Open Documents event.

At the first line, AEGetParamDesc is called to get the direct parameter (specified in the keyDirectObject keyword) out of the Apple event. The constant typeAEList specifies the descriptor type as a list of descriptor structures. The descriptor list is received by the docList variable.

Before proceeding further, the handler checks that it has received all the required parameters by calling the function doHasGotRequiredParams.

Having retrieved the descriptor list from the Apple event, the handler calls AECountItems to count the number of descriptors in the list.

Using the returned number as an index, AEGetNthPtr is called to get the data of each descriptor structure in the list. In the AEGetNthPtr call, the parameter typeFSS specifies the desired type of the resulting data, causing the Apple Event Manager to coerce the data in the descriptor structure to a file system specification structure. Note also that keyWord receives the keyword of the specified descriptor structure, returnedType receives the descriptor type, fileSpec receives a pointer to the file system specification structure, sizeof(fileSpec) establishes the length, in bytes, of the data returned, and actualSize receives the actual length, in bytes, of the data for the descriptor structure.

After extracting the file system specification structure describing the document to open, the handler calls the function for opening files (doOpenFile). (In a real application, that function would typically be the same as that invoked when the user chooses Open from the application's File menu.)

If the call to AEGetNthPtr does not return noErr, the error handling function (doErrorAlert) is called. (AEGetNthPtr will return an error code if there was insufficient room in the heap, the data could not be coerced, the descriptor structure was not found, the descriptor was of the wrong type or the descriptor structure was not a valid descriptor structure.)

If the call to doHasGotRequiredParams does not return noErr, the error handling function (doErrorAlert) is called. (doHasGotRequiredParams returns noErr only if you got all the required parameters.)

Since the handler has no further requirement for the data in the descriptor list, AEDisposeDesc is called to dispose of the descriptor list.

If the call to AEGetParamDesc does not return noErr the error handling function (doErrorAlert) is called. (AEGetParamDesc will return an error code for much the same reasons as will AEGetNthPtr.)

doPrintDocsEvent

doPrintDocsEvent is the handler for the Print Documents event.

The code is identical to that for the Open Documents event handler doOpenDocs except that the function for printing files (doPrintFile) is called rather than the function for simply opening files (doOpenFile).

doQuitAppEvent

doQuitAppEvent is the handler for the Quit Application event.

After checking that it has received all the required parameters by calling the function doHasGotRequiredParams, the handler calls the function doPrepareToTerminate.

doSysFontChangeEvent, doSmallSysFontChange, and doViewsFontChangeEvent

doSysFontChangeEvent, doSmallSysFontChange, and doViewsFontChangeEvent are the handlers for the appearance Apple events.

The function doHasGotRequiredParams is called to check whether the Apple event contains any required parameters. If so, the handler returns an error because, by definition, these events should not contain any required parameters.

The handlers then draw some advisory text in the non-document window indicating that the event has been received, call the Appearance Manager function GetThemeFont to obtain information about the relevant font (large system, small system, or views), and draw the font name (and, in the case of the views font, the font size).

showPreferencesEventHandler

showPreferencesEventHandler is the handler for the Show Preferences Apple event. It simply draws some advosory text in the window to prove that the event was received.

doHasGotRequiredParams

doHasGotRequiredParams is the function called by doOpenAppEvent, doReopenAppEvent, doSysFontChangeEvent, doSmallSysFontChange, and doViewsFontChangeEvent to confirm that the event passed to it contains no required parameters, and by the other required Apple event handlers to check that they have received all the required parameters.

The first parameter in the call to AEGetAttributePtr is a pointer to the Apple event in question. The second parameter is the Apple event keyword; in this case the constant keyMissedKeywordAttr is specified, meaning the first required parameter remaining in the event. The third parameter specifies the descriptor type; in this case the constant typeWildCard is specified, meaning any descriptor type. The fourth parameter receives the descriptor type of the returned data. The fifth parameter is a pointer to the data buffer which stores the returned data. The sixth parameter is the maximum length of the data buffer to be returned. Since we do not need the data itself, these parameters are set to NULL and 0 respectively. The last parameter receives the actual length, in bytes, of the data buffer for the attribute.

AEGetAttributePtr returns the result code errAEDescNotFound if the specified descriptor type (typeWildCard, that is, any descriptor type) is not found, meaning that the handler extracted all the required parameters. In this event, doHasGotRequiredParams returns noErr.

If AEGetAttributePtr returns noErr, the handler has not extracted all of the required parameters, in which case, the handler should return errAEParamMissed and not handle the event. Accordingly, errAEParamMissed is returned to the handler (and, in turn, by the handler) if noErr is returned by AEGetAttributePtr.

doOpenFile

doOpenFile takes the file system specification structure and opens a window with the filename contained in that structure repeated in the window's title bar (the calls to doNewWindow and SetWTitle). The rest of the doOpenFile code simply draws explanatory text in the text window.

In a real application, this is the function that would open files as a result of, firstly, the receipt of the Open Documents event and, secondly, the user choosing Open from the application's File menu and then choosing a file or files from the resulting Navigation Services Open dialog.

doPrintFile

doPrintFile is applicable to Mac OS 8/9 only. It is the function which, in a real application, would take the file system specification structure passed to it from the Print Documents event handler, extract the filename and control the printing of that file. In this demonstration, most of the doPrintFile code is related to drawing explanatory text in the text window.

If your application can interact with the user, it should open windows for the documents, display a Print dialog for the first document, and use the settings entered by the user for the first document to print all documents.

Note that, if your application was not running when the user selected a document icon and chose Print from the Finder's File menu, the Finder will send a Quit Application event following the print operation.

doPrepareToTerminate

doPrepareToTerminate is the function called by the Quit Application event handler. In this demonstration, gDone will be set to true, and the program will thus terminate immediately, if the Quit Application event

resulted from the user initiating a print operation from the Mac OS 8/9 Finder when the application was not running.

If the application was running (gApplicationWasOpen contains true) and the Quit Application event thus arose from the user selecting Restart or Shut Down from the Finder's File menu, the demonstration waits for a button click before setting gDone to true. (In a real application, and where appropriate, this area of the code would invoke alerts to ascertain whether the user wished to save changed documents before closing down.)

Note that, when your application is ready to quit, it must call ExitToShell from the main event loop, not from the handlers for the Quit Application event. Your application should quit only after the handler returns noErr as its function result.

doNewWindow

doNewWindow opens document windows in response to calls from the Open Application and Open Documents event handlers.