

10

APPLE EVENTS

Includes Demonstration Program AppleEvents

Introduction

As stated at Chapter 2 — Low Level and Operating System Events, events are broadly categorised as low-level events, Operating System events, and **high-level** events.

Using high-level events, an application can instruct another application to perform a specific action, such as adding a row to a spreadsheet or changing the font size of a paragraph. An application can also request information from another application; for example, it might request a dictionary application to return the definition of a particular word.

Fig 1 shows the general event-handling mechanism. In Fig 1, three different applications are communicating with each other by sending and receiving high-level events. Note that high-level events are placed in a separate queue maintained by the operating system and that a high-level event queue is maintained for each application that has announced itself as capable of receiving high-level events.

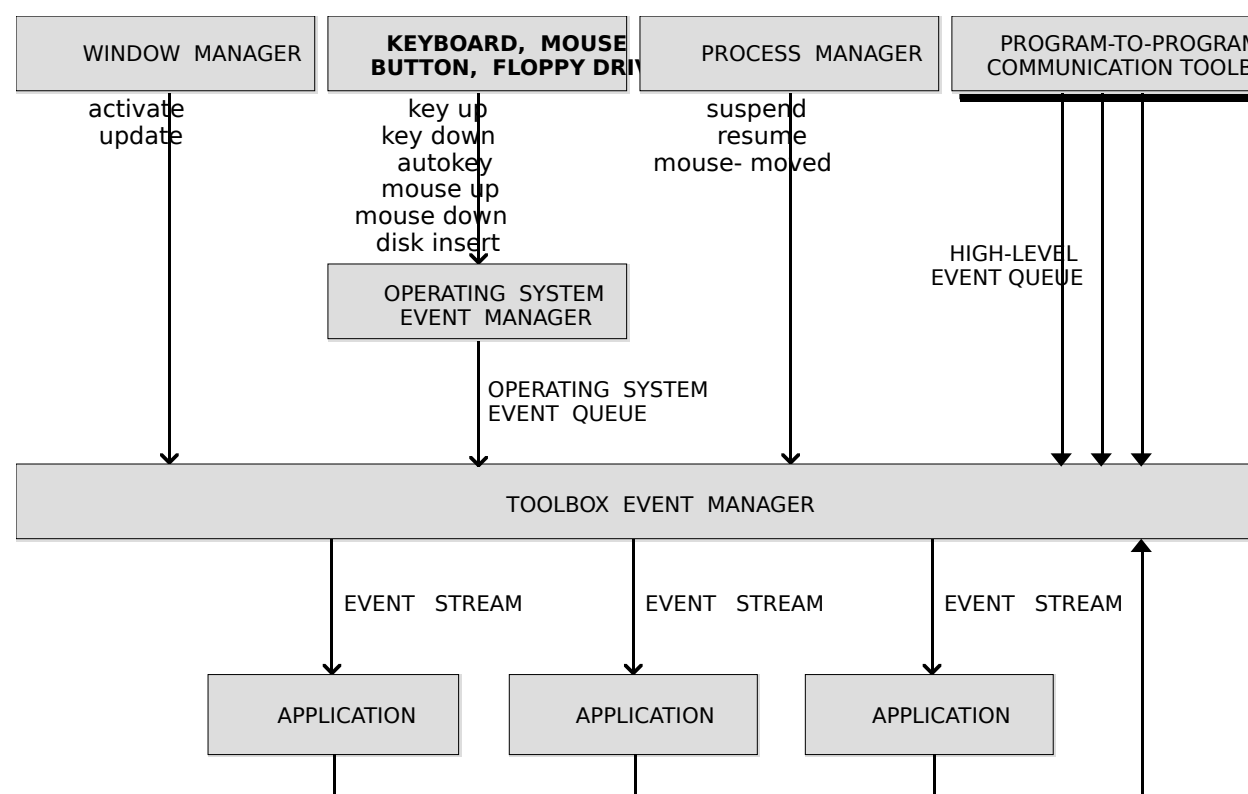


FIG 1 - GENERAL EVENT HANDLING MECHANISM

For effective communication between applications, an application must define the set of high-level events it responds to and let other applications know the events it accepts. For a high-level event sent by one application to be understood by another application, the sender and receiver must agree on a **protocol**, that is, on the way the event is to be interpreted.

Apple Events

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.

Communication between two applications which support Apple events is initiated by a **client application**, which sends an Apple event to request a service or information. The application providing the service or information is called a **server application**.¹ Fig 2 shows a common Apple event, called the Open Documents event. The Finder (which is, itself, an application) is the client; it requests that the application My Application open the documents named Document A and Document B. My Application responds by opening windows for the specified documents.

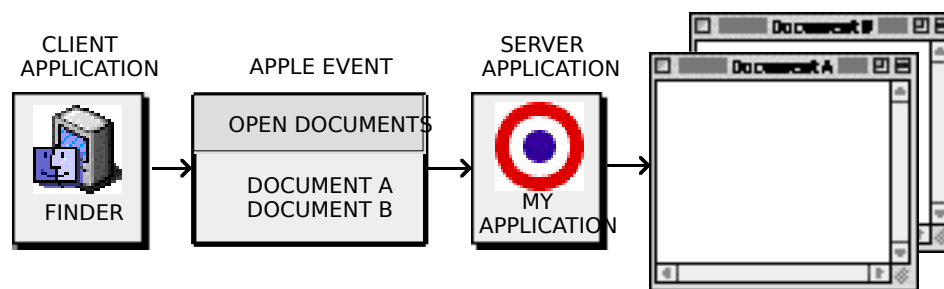


FIG 2 - CLIENT AND SERVER

To identify Apple events and respond appropriately, every application can rely on a vocabulary of standard Apple events which developers and Apple have established for all applications to use. These events are defined in the Apple Event Registry: Standard Suites. The standard **suites** (groups 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.

Historical Note

The Re-open Application Apple event was introduced with Mac OS 8.

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.

¹ An application can also send Apple events to itself, thus acting as both client and server.

- 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 which support a related functional area, and which include the Text suite and the Database suite.
- The **appearance** suite, which consists of four Apple events used to advise all foreground applications when one of the following changes has been made in the Fonts tab of the Appearance control panel:
 - Appearance.
 - Large system font.
 - Small system font.
 - Views font.

Historical Note

The Apple event relating to an appearance change was introduced with Mac OS 8. The Apple events relating to large system font, small system font, and views font changes were introduced with Mac OS 8.5.

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

Apple Event Attributes and Parameters

When an application creates and sends an Apple event, the Apple Event Manager uses arguments passed to Apple Event Manager functions to construct the data structures that make up the Apple event. An Apple event comprises **attributes** (which identify the Apple event and denote its task) and, often, **parameters** (which contain information to be used by the target application).

Apple Event Attributes

An Apple event attribute is a structure which identifies the **event class**, **event ID**, target application, and other characteristics of an Apple event. Taken together, the attributes denote the task to be performed on any data specified in the event's parameters. After receiving an Apple event, a server application can use Apple Event Manager functions to extract and examine its attributes. Apple events are identified by their event class and event ID attributes.

Event Class

The event class is the attribute that identifies a group of related Apple events. It appears in the `message` field of the event structure for an Apple event (see Fig 3). 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. 'aevt' is represented by the constant `kAppearanceEventClass`.

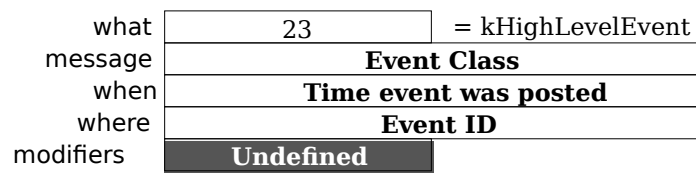


FIG 3 - CONTENTS OF AN EVENT STRUCTURE - HIGH LEVEL

Event ID

The event ID is the attribute which identifies the particular event within the event class. In conjunction with the event class, the event ID uniquely identifies the Apple event and communicates what action the Apple event should perform. It appears in the `where` field of the event structure for an Apple event (see Fig 3). For example, the event ID of an Open Documents event has the value 'odoc', which is represented by the constant `kAEOpenDocuments`. The `kCoreEventClass` constant in combination with the `kAEOpenDocuments` constant identifies the Open Documents event to the Apple Event Manager.

The following are the event IDs for the five required Apple events and the four Appearance Manager Apple events. (Those introduced with Mac OS 8.5 appear on a dark gray background.)

Event ID	Value	Description
<code>kAEOpenApplication</code>	'oapp'	Perform tasks required when a user opens your application.
<code>kAEReopenApplication</code>	'rapp'	Perform tasks required when a user "re-opens" your application.
<code>kAEOpenDocuments</code>	'odoc'	Open documents.
<code>kAEPrintDocuemnts</code>	'pdoc'	Print Documents.
<code>kAEQuitApplication</code>	'quit'	Quit your application.
<code>kAEAppearanceChanged</code>	'thme'	Current appearance has changed. Action as required.
<code>kAESystemFontChanged</code>	'sysf'	Current system font has changed. Action as required.
<code>kAESmallSystemFontChanged</code>	'ssfnt'	Current small system font has changed. Action as required.
<code>kAEViewsFontChanged</code>	'vfnt'	Current views font has changed. Action as required.

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 that the target application uses. 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. As with attributes, a client application can use Apple Event Manager functions to extract and examine the parameters of an Apple event it has received.

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

Direct Parameters

A direct parameter usually specifies the data to be acted upon by the target application. For example, a list of documents is contained in the direct parameter of the Print Documents event.

Additional Parameters

Some Apple events also take additional parameters, which the target application uses in addition to the data specified in the direct parameter. For example, an Apple event for arithmetic operations may include additional parameters which specify operands in an equation.

Required and Optional Parameters

All parameters are either **required parameters** or **optional parameters**. A required parameter is one which must be present for the target application to carry out the task denoted by the Apple event. An optional parameter is a supplemental Apple event parameter that can also be used to specify data to a target application. Direct parameters are usually defined as required parameters in the Apple Event Registry - Standard Suites.

Interpreting Apple Event Attributes and Parameters

Fig 4 shows the major Apple event attributes and direct parameter for the Open Documents event.

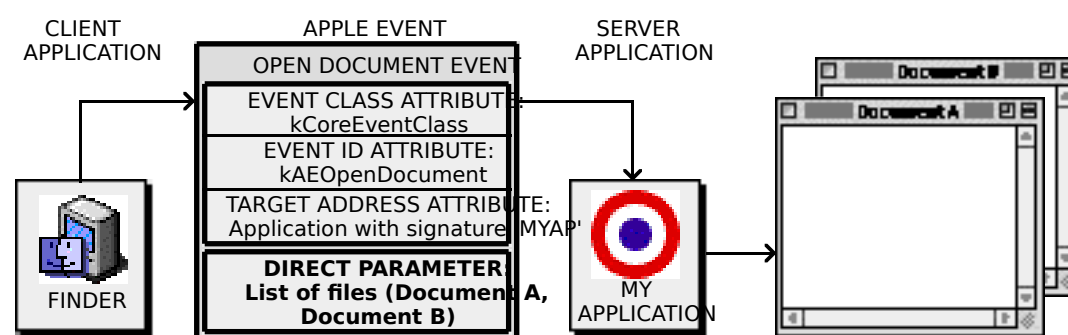


FIG 4 - MAJOR ATTRIBUTES AND DIRECT PARAMETERS IN AN OPEN I

To process this event, your application would use the `AEProcessAppleEvent` function, which uses the event class and event ID attributes to dispatch the event to My Application's Open Documents handler. In response, 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

Descriptor structures are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A **descriptor structure** is a data structure of type `AEDesc`. It consists of a handle to data and a descriptor type which identifies the type of data to which the handle refers:

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

The **descriptor type** is a structure of type `DescType` which, in turn, is of data type `ResType`, that is, a four-character code. Constants are used in place of these codes when referring

to descriptor types. The following are some of the major descriptor type constants, their values, 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).

The following illustrates the logical arrangement of a descriptor structure with a descriptor of type `typeChar`, which specifies that the data handle refers to an unterminated string:

Data Type AEDesc

Descriptor type:	typeChar
Data:	"Summary of Sales"

The following illustrates the logical arrangement of a descriptor structure with a descriptor type of `typeType`, which specifies that the data handle refers to a four-character code (in this case the constant `kCoreEventClass`, whose value is 'aevt'):

Data Type AEDesc

Descriptor type:	typeType
Data:	(kCoreEventClass)

Address Descriptor Structure

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.
```

The address in an address descriptor structure can be specified as one of the four basic types (or as any other descriptor types you define that can be coerced to one of these types):

Descriptor Type	Value	Description
typeApplSignature	'sign'	Application signature.
typeSessionID	'ssid'	Session reference number.
typeTargetID	'targ'	Target ID structure.
typeProcessSerialNumber	'psn '	Process serial number.

Like several other data structures defined by the Apple Event Manager for use in Apple event attributes and Apple event parameters, an address descriptor structure is identical

to a descriptor structure of data type `AEDesc`; the only difference is that the data for an address descriptor structure must always consist of an application's address.

Keyword-Specified Descriptor Structures

After the Apple Event Manager has assembled the necessary descriptor structures as the attributes and parameters of an Apple event, your application must use Event Manager functions to request each attribute and parameter by **keyword**. Keywords are arbitrary names used by the Apple Event Manager to keep track of various descriptor structures. The `AEKeyword` data type is defined as a four-character code:

```
typedef FourCharCode AEKeyword;
```

Constants are typically used to represent keywords.

Keywords for Attributes. The following is a partial list of keyword constants for Apple event attributes:

Attribute Keyword	Value	Description
<code>keyEventClassAttr</code>	'evcl'	Event class of Apple event.
<code>keyMissedKeywordAttr</code>	'miss'	Keyword for first required parameter remaining in an Apple event.
<code>keyAddressAttr</code>	'addr'	Address of target or client application.
<code>keyEventIDAttr</code>	'evid'	Event ID of Apple event.
<code>keyEventSourceAttr</code>	'esrc'	Nature of the source application.
<code>keyReturnIDAttr</code>	'rtid'	Return ID for reply Apple event.

Keywords for Parameters. The following is a list of keyword constants for commonly used Apple event parameters:

Parameter Keyword	Value	Description
<code>keyDirectObject</code>	'----'	Direct parameter.
<code>keyErrorNumber</code>	'errn'	Error number parameter.
<code>keyErrorString</code>	'errs'	Error string parameter.

The Apple Event Manager associates keywords with specific descriptor structures by means of a **keyword-specified descriptor structure**, a data structure of type `AEKeyDesc` that consists of a keyword and a descriptor structure:

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

The following illustrates a keyword-specified descriptor structure with the keyword `keyEventClassAttr`, the keyword that identifies an event class attribute. It shows the logical arrangement of the event class attribute for the Open Documents event shown at Fig 4.

Data Type `AEKeyDesc`

Keyword:	keyEventClassAttr	
Descriptor Structure:	Descriptor Type:	typeType
	Data:	Event Class (coreEventClass)

Descriptor Lists, AE Structures, and AppleEvent Structures

Descriptor Lists

When extracting data from an Apple event, you use Apple Event Manager functions to copy data to a buffer specified by a pointer, or to return a descriptor structure whose data handle refers to a copy of the data, or to return lists of descriptor structures (called **descriptor lists**).

A descriptor list is a data structure of type `AEDescList` defined by the data type `AEDesc`. That is, 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.
```

The following illustrates the logical arrangement of the descriptor list that specifies the direct parameter of the Open Documents event shown at Fig 4. This descriptor list consists of a list of descriptor structures which contain alias structures to filenames.

Data Type AEDescList

Descriptor type:	typeAEList	
Data:	List of descriptor structures:	
	Descriptor type:	typeAlias
	Data:	Alias structure for filename (Document A)
	Descriptor type:	typeAlias
	Data:	Alias structure for filename (Document B)

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

AE Structure

Keyword-specified descriptor structures for Apple event parameters can in turn be combined into an **AE structure**, which is a descriptor list of type `AERecord`:

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

The handle for a descriptor list of data type `AERecord` refers to a list of keyword-specified descriptor structures that can be used to construct Apple event parameters. An AE structure has the descriptor type `typeAERecord` and can be coerced to several other descriptor types.

Apple Event Structure

An **Apple event structure**, which is different from an AE structure, is another special descriptor list of data type `AppleEvent` and descriptor type `typeAppleEvent`:

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

An Apple event structure describes a full-fledged Apple event. Like the data for an AE structure, the data for an Apple event structure consists of a list of keyword-specified descriptor structures. Unlike an AE structure, the data for an Apple event structure is divided into two parts, one for attributes and one for parameters. This division allows the Apple event to distinguish between an Apple event's attributes and its parameters.

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

Descriptor lists, AE structures and Apple event structures are all descriptor structures whose handles refer to a nested list of other descriptor structures. The data associated with each data type may be organised differently and used by the Apple Event Manager for different purposes. In each case, however, the data are identified by a handle in a descriptor structure. This means that you can pass an Apple event structure to any Apple Event Manager function that expects an AE structure. Similarly, 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 5 shows an example of a complete Apple event — a data structure of type `AppleEvent` containing a list of keyword-specified descriptor structures that name the attributes and parameters of an Open Documents event.

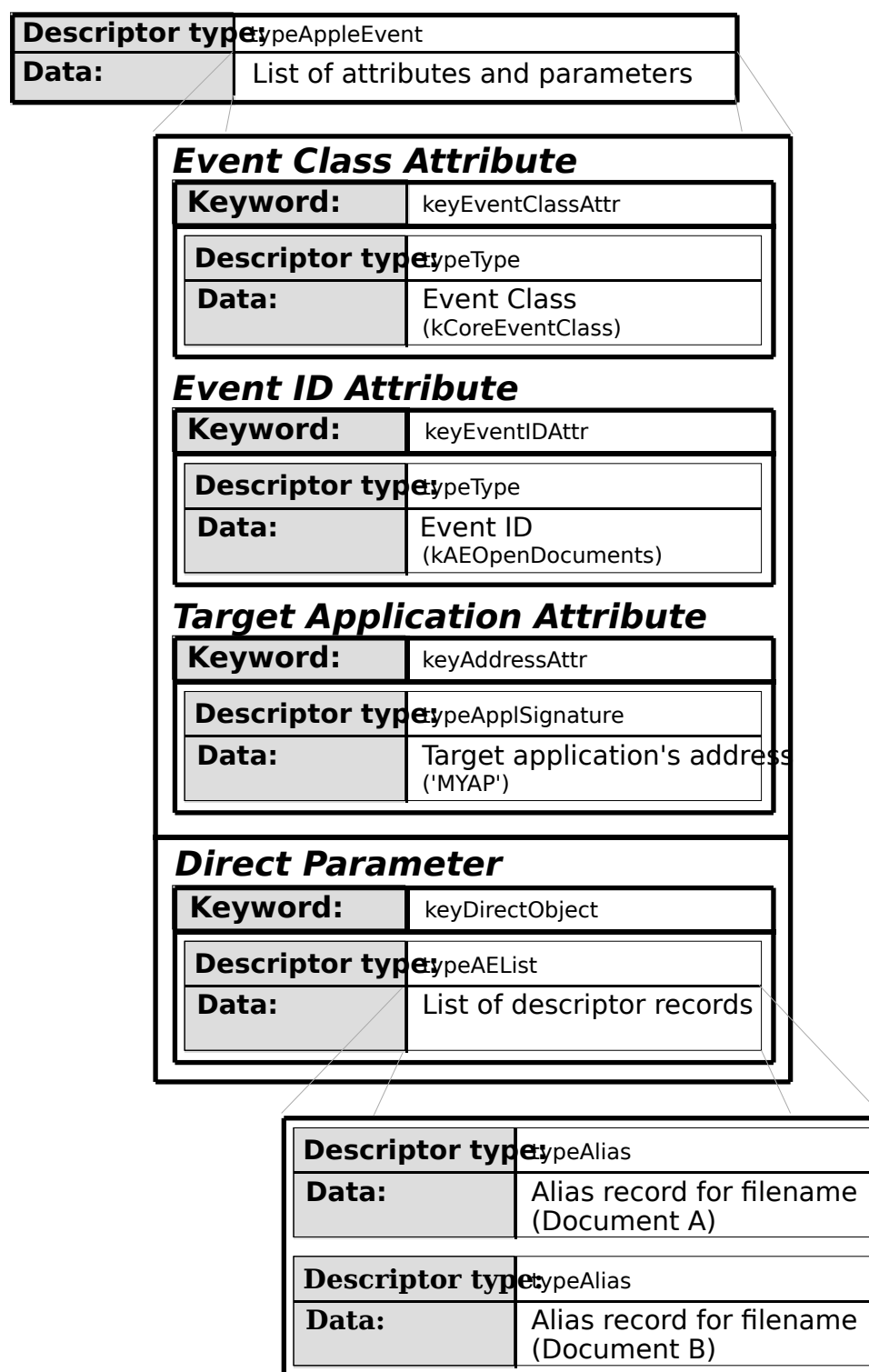


FIG 5 - DATA STRUCTURES WITHIN AN OPEN DOCUMENT

Handling Apple Events

A client application uses the Apple Event Manager to create and send an Apple event requesting a service or information. A server application responds by using the Apple Event Manager to process the Apple event, extract data from the attributes and parameters of the Apple event and, if necessary, add requested data to the reply event returned by the Apple Event Manager to the client application.

As a first step in supporting Apple events, and as previously stated, your application should support the required Apple events sent by the Finder. To support the required Apple events, you must:

- Set the `isHighLevelEventAware` flag in the 'SIZE' resource of your application.

- Test for high-level events in your application's 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. Your application should therefore test the `what` field of the event structure to determine whether it contains the value represented by `kHighLevelEvent`.
- Use `AEProcessAppleEvent` to process the Apple events. `AEProcessAppleEvent` first identifies the Apple event by examining the data in the event class and event ID attributes. It then uses that data to call 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. This function installs handlers in an **Apple event dispatch table** for your application. The Apple Event Manager uses this table to map Apple events to handlers in your application. When your application calls `AEProcessAppleEvent`, the Apple Event Manager checks the dispatch table and, if your application has installed a handler for the event, calls the handler. Each entry in the Apple event dispatch table should specify:
 - The event class.
 - The event ID.
 - A universal procedure pointer to the Apple event handler.
 - A reference constant.²

Accordingly, the parameters for the call to `AEInstallEventHandler` are the event class, the event ID, a pointer to the event handler, a reference constant, and `false`.³

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 parameters and attributes from the Apple event.
- Check that all required parameters have been extracted.

² The reference constant is passed to your handler by the Apple Event Manager each time your handler is called. Your application can use this reference constant for any purpose. If your application does not use the reference constant, specify 0.

³ `false` causes the handler to be installed in the application's Apple event dispatch table. `true` causes the handler 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. The handlers in your application's Apple events dispatch table are available only to your application. If `AEProcessAppleEvent` cannot find a handler for the Apple event in your application's Apple event dispatch table, it looks in the system Apple event dispatch table for a handler. If it does not find a handler in the system table, it returns the `errAEventNotHandled` result code.

- 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.
AECntItems	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.

Interacting With the User

In some cases, the server application 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 .

The Apple Event Manager does not allow the server application to interact with the user in response to a client application's Apple event unless at least two conditions are met:

- First, the client application must set flags in the `sendMode` parameter of the `AESend` function to indicate that user interaction is allowed.
- Second, the server application must either:
 - Set flags to the `AESetInterActionAllowed` function indicating that user interaction is allowed. (These flags relate to permitting interaction where the client and

server are the same application, the client application is on the same computer as the server, or the client is on any computer.)

- Set no user interaction preferences (that is, make no call to `AESetInterActionAllowed`), in which case `AEInteractWithUser` (the function used to initiate interaction with the user when your application is a server responding to an Apple event) assumes that only interaction with a client on the local computer is allowed.

If these two conditions are met, and if `AEInteractWithUser` determines that both the client and server applications allow user interaction under the current circumstances, `AEInteractWithUser` brings your application to the foreground if it is not already in the foreground. Your application can then display its dialog box or alert box or otherwise interact with the user.

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. If your handler returns a non-zero result code, the Apple Event Manager adds a `keyErrorNumber` parameter to the reply Apple event. This parameter contains the result code that your handler returns.

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 or by selecting the icon and choosing Open from the Finder's File menu), the Finder launches your application (using the Process Manager) and sends it an Open Application event.
- If your application is already open and the user attempts to "open" it from the Finder (either by double clicking the application's icon or by selecting the icon and choosing Open from the Finder's File menu), the Finder sends your application a Re-open Application event.⁴
- If your application is not open and the user elects to open one of your application's documents from the Finder, the Finder launches your application (using the Process Manager) and sends it an Open Documents event.
- If your application is not open and the user elects to print one of your application's documents from the Finder, the Finder launches your application (using the Process Manager) and sends it the Print Documents event. Your application should print the

⁴ 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.

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 print any of your application's documents from the Finder, the Finder sends your application the Open Documents or Print Documents event.
- If your application is open and the user chooses Restart or Shut Down from the Finder's Special menu, 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, opening windows for the documents only if your application can interact with the user.

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 requesting the user 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 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

In the following, those constants introduced with Mac OS 8.5 appear on a dark gray background.

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

kAEAppearanceChanged = FOUR_CHAR_CODE('thme') // Appearance changed.
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.

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.
errAEParmMissed       = -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 of 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.
    Handle    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 AEGgetParamDesc(const AppleEvent *theAppleEvent,AEKeyword theAEKeyword,
    DescType desiredType,AEDesc *result);
OSErr AEGgetAttributePtr(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 AEGgetNthPtr(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);
```

Accessing Theme Font Information

```
OSStatus   GetThemeFont(ThemeFontID inFontID,ScriptCode inScript,StringPtr outFontName,  
                        Sint16 *outFontSize, Style *outStyle);  
OSStatus   UseThemeFont(ThemeFontID inFontID,ScriptCode inScript);
```



```

//
..... includes

#include <Appearance.h>
#include <AERegistry.h>
#include <Devices.h>
#include <Dialogs.h>
#include <Gestalt.h>
#include <ToolUtils.h>

//
..... defines

#define rMenubar          128
#define mApple           128
#define mFile            129
#define iQuit            11
#define rDisplayWindow   128
#define rDocWindow       129
#define rErrorStrings    128
#define eInstallHandler  1
#define eGetRequiredParam 2
#define eGetDescriptorRecord 3
#define eMissedRequiredParam 4
#define eCannotOpenFile  5
#define eCannotPrintFile 6
#define eCannotOpenWindow 7
#define eMenus            8
#define MIN(a,b)         ((a) < (b) ? (a) : (b))

//
..... global variables

AEEEventHandlerUPP doOpenAppEventUPP;
AEEEventHandlerUPP doReopenAppEventUPP;
AEEEventHandlerUPP doOpenDocsEventUPP;
AEEEventHandlerUPP doPrintDocsEventUPP;
AEEEventHandlerUPP doQuitAppEventUPP;

AEEEventHandlerUPP doAppearanceChangeUPP;
AEEEventHandlerUPP doSysFontChangeUPP;
AEEEventHandlerUPP doSmallSysFontChangeUPP;
AEEEventHandlerUPP doViewsFontChangeUPP;

Boolean    gMacOS85Present    = false;
WindowPtr  gWindowPtr;
Boolean    gDone;
Boolean    gApplicationWasOpen = false;
WindowPtr  gWindowPtrs[10];
SInt16     gNumberOfWindows   = 0;

//
..... function prototypes

void        main                (void);
void        doInitManagers      (void);
void        doInstallAEHandlers (void);
void        doEvents            (EventRecord *);
pascal OSErr doOpenAppEvent     (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doReopenAppEvent   (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doOpenDocsEvent    (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doPrintDocsEvent   (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doQuitAppEvent     (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doAppearanceChangeEvent (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doSysFontChangeEvent (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doSmallSysFontChangeEvent (AppleEvent *,AppleEvent *,SInt32);
pascal OSErr doViewsFontChangeEvent (AppleEvent *,AppleEvent *,SInt32);
OSErr      doHasGotRequiredParams (AppleEvent *);
Boolean    doOpenFile           (FSSpec *,SInt32,SInt32);
Boolean    doPrintFile          (FSSpec *,SInt32,SInt32);
void       doPrepareToTerminate (void);
WindowPtr  doNewWindow          (void);
void       doMenuChoice         (SInt32);
void       doErrorAlert         (SInt16);
void       doDrawText           (Str255);

```


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 or resulted from the user choosing Restart or Shut Down from the Finder's Special menu.

gWindowPtrs will be assigned pointers to the document windows. gNumberOfWindows is used to increment the gWindowPtrs[] array element after each document window is created.

main

The main function initialises the system software managers and then determines whether Mac OS 8.5 is present.

The next block calls NewAEventHandlerProc nine times to create routine descriptors for each of the required Apple event and appearance Apple event handling functions. Note that the last three calls are made only if Mac OS 8.5 or later is present; this is because the Mac OS 8.5 Appearance control panel was the first to contain pop-up menus allowing the user to change the system, small system, and views fonts.

The next block calls GetNewCWindow to open the text display window, makes that window the current graphics port, sets the text size for that graphics port to 10pt, sets the text style to bold, sets the foreground and background colours, and calls EraseRect to draw the window's content area in the new background colour.

The menus are then set up. Note that here, and in other areas of the program, an error will cause the application-defined error-handling function doErrorAlert to be called.

The call to doInstallAEHandlers installs the Apple event handlers. Finally, the main event loop is entered.

doInstallAEHandlers

doInstallAEHandlers installs the handlers for the required Apple events and the Appearance Manager Apple events in the application's Apple event dispatch table. Note that the last three are only installed if Mac OS 8.5 or later is present.

doEvents

doEvents switches according to the event type received.

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.)

doMouseDown

doMouseDown performs such mouse-down processing as is necessary to support the demonstration aspects of the program.

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 application-defined 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.

The last two lines mean that, if errAEParmMissed 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

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

At the first line, the application-defined 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.

The last two lines mean that, if errAEParmMissed 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, `AEGgetParamDesc` 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 application-defined function `doHasGotRequiredParams` (Line 324).

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, `AEGGetNthPtr` is called to get the data of each descriptor structure in the list. In the `AEGGetNthPtr` 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 application-defined 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 `AEGGetNthPtr` does not return `noErr`, the application-defined error handling function (`doErrorAlert`) is called. (`AEGGetNthPtr` 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 application-defined 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 `AEGgetParamDesc` does not return `noErr` the application-defined error handling function (`doErrorAlert`) is called. (`AEGgetParamDesc` will return an error code for much the same reasons as will `AEGGetNthPtr`.)

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 application-defined 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 application-defined function `doHasGotRequiredParams`, the handler calls the application-defined function `doPrepareToTerminate`.

doAppearanceChangeEvent, doSysFontChangeEvent, doSmallSysFontChange, and doViewsFontChangeEvent

`doAppearanceChangeEvent`, `doSysFontChangeEvent`, `doSmallSysFontChange`, and `doViewsFontChangeEvent` are the handlers for the Appearance Manager Apple events.

After checking that they have received all the required parameters by calling the application-defined function `doHasGotRequiredParams`, the handlers 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).

doHasGotRequiredParams

doHasGotRequiredParams is the application-defined function called by doOpenAppEvent and doThemeSwitch 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 AEGetAddressPtr 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.

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

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 Open dialog box.

doPrintFile

doPrintFile 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 Job 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 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 dialog boxes to ascertain whether the user wished to save any 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.

doMenuChoice, doErrorAlert, doDrawText, and doConcatPStrings

doMenuChoice handles menu selections. gDone is set to true when the user selects Quit from the application's File menu. doErrorAlert handles errors, displaying a movable modal alert box with descriptive text and, where necessary, terminating the program. doDrawText draws scrolling explanatory text in the text window as each event is received. DoConcatPStrings concatenates two Pascal strings.