Apple Event Manager Reference

Carbon > Events & Other Input

2007-07-13

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Simultaneously published in the United States and Canada.

Contents

Apple Event Manager Reference

Framework:	CoreServices/CoreServices.h, Carbon/Carbon.h
Declared in	AEDataModel.h AEHelpers.h AEInteraction.h AEMach.h AEObjects.h AEPackObject.h AERegistry.h AEUserTermTypes.h AppleEvents.h

Overview

The Apple Event Manager, a part of the Open Scripting Architecture (OSA), provides facilities for applications to send and respond to Apple events and to make their operations and data available to AppleScript scripts. For related API reference, see Open Scripting Architecture Reference.

An Apple event is a type of interprocess message that can specify complex operations and data. Apple events provide a data transport and event dispatching mechanism that can be used within a single application, between applications on the same computer, and between applications on different computers connected to a network.

Applications typically use Apple events to request services and information from other applications or to provide services and information in response to such requests. All applications that present a graphical interface to the user through the Human Interface Toolbox (Carbon applications) or the Cocoa application framework should be able to respond, if appropriate, to certain events sent by the Mac OS. These include the `open application` (or `launch`), `reopen`, `open documents`, `print documents`, and `quit` events.

Some Apple Event Manager functions are marked as being thread safe—for all other functions, you should call them only on the main thread.

For an overview of technologies that take advantage of the Apple Event Manager, see AppleScript Overview.

For information on working with Apple events, including events sent by the Mac OS, see "Responding to Apple Events" in Apple Events Programming Guide. For information about individual four-character codes used in Apple events, see *AppleScript Terminology and Apple Event Codes Reference*.

The Apple Event Manager is implemented by the AE framework, a subframework of the Core Services framework. You don't link directly with the AE framework—instead, you typically link with the Carbon framework, which includes it. Some AppleEvent definitions are only available to clients of the Carbon framework, which includes, for example, `AEInteraction.h` in the HIToolbox framework.

The AE framework does not force a connection to the window server. This allows daemons and startup items that work with Apple events to continue working across log outs.

Functions by Task

Adding Items to Descriptor Lists

`AEPutArray` (page 75)
: Inserts the data for an Apple event array into a descriptor list, replacing any previous descriptors in the list.

`AEPutDesc` (page 77)
: Adds a descriptor to any descriptor list, possibly replacing an existing descriptor in the list.

`AEPutPtr` (page 81)
: Inserts data specified in a buffer into a descriptor list as a descriptor, possibly replacing an existing descriptor in the list.

Adding Parameters and Attributes to Apple Events and Apple Event Records

`AEPutAttributeDesc` (page 76)
: Adds a descriptor and a keyword to an Apple event as an attribute.

`AEPutAttributePtr` (page 77)
: Adds a pointer to data, a descriptor type, and a keyword to an Apple event as an attribute.

`AEPutKeyDesc` (page 78)
: Inserts a descriptor and a keyword into an Apple event record as an Apple event parameter.

`AEPutKeyPtr` (page 79)
: Inserts data, a descriptor type, and a keyword into an Apple event record as an Apple event parameter.

`AEPutParamDesc` (page 80)
: Inserts a descriptor and a keyword into an Apple event or Apple event record as an Apple event parameter.

`AEPutParamPtr` (page 80)
: Inserts data, a descriptor type, and a keyword into an Apple event or Apple event record as an Apple event parameter.

Coercing Descriptor Types

`AECoerceDesc` (page 29)
: Coerces the data in a descriptor to another descriptor type and creates a descriptor containing the newly coerced data.

`AECoercePtr` (page 30)
: Coerces data to a desired descriptor type and creates a descriptor containing the newly coerced data.

Counting the Items in Descriptor Lists

`AECountItems` (page 31)
Counts the number of descriptors in a descriptor list.

Creating an Apple Event

`AECreateAppleEvent` (page 32)
Creates an Apple event with several important attributes but no parameters.

Creating and Duplicating Descriptors

`AECreateDesc` (page 33)
Creates a new descriptor that incorporates the specified data.

`AECreateDescFromExternalPtr` (page 34)
Creates a new descriptor that uses a memory buffer supplied by the caller.

`AEDuplicateDesc` (page 42)
Creates a copy of a descriptor.

Creating, Calling, and Deleting Universal Procedure Pointers

`DisposeAECoerceDescUPP` (page 119)
Disposes of a universal procedure pointer to a function that coerces data stored in a descriptor.

`DisposeAECoercePtrUPP` (page 119)
Disposes of a universal procedure pointer to a function that coerces data stored in a buffer.

`DisposeAEDisposeExternalUPP` (page 119)
Disposes of a universal procedure pointer to a function that disposes of data supplied to the `AECreateDescFromExternalPtr` function.

`DisposeAEEventHandlerUPP` (page 120)
Disposes of a universal procedure pointer to an event handler function.

`DisposeAEFilterUPP` (page 120)
Disposes of a universal procedure pointer to an Apple event filter function.

`DisposeAEIdleUPP` (page 120)
Disposes of a universal procedure pointer to an Apple event idle function.

`DisposeOSLAccessorUPP` (page 120)
Disposes of a universal procedure pointer to an object accessor function.

`DisposeOSLAdjustMarksUPP` (page 121)
Disposes of a universal procedure pointer to an object callback adjust marks function.

`DisposeOSLCompareUPP` (page 121)
Disposes of a universal procedure pointer to an object callback comparison function.

`DisposeOSLCountUPP` (page 121)
Disposes of a universal procedure pointer to an object callback count function.

`NewAEFilterUPP` (page 131)
Creates a new universal procedure pointer to an Apple event filter function.

`NewAEIdleUPP` (page 131)
Creates a new universal procedure pointer to an Apple event idle function.

`NewOSLAccessorUPP` (page 132)
Creates a new universal procedure pointer to an object accessor function.

`NewOSLAdjustMarksUPP` (page 132)
Creates a new universal procedure pointer to an object callback adjust marks function.

`NewOSLCompareUPP` (page 133)
Creates a new universal procedure pointer to an object callback comparison function.

`NewOSLCountUPP` (page 133)
Creates a new universal procedure pointer to an object callback count function.

`NewOSLDisposeTokenUPP` (page 133)
Creates a new universal procedure pointer to an object callback dispose token function.

`NewOSLGetErrDescUPP` (page 134)
Creates a new universal procedure pointer to an object callback get error descriptor function.

`NewOSLGetMarkTokenUPP` (page 134)
Creates a new universal procedure pointer to an object callback get mark function.

`NewOSLMarkUPP` (page 134)
Creates a new universal procedure pointer to an object callback mark function.

Creating Descriptor Lists and Apple Event Records

`AECreateList` (page 35)
Creates an empty descriptor list or Apple event record.

Creating Object Specifiers

`CreateCompDescriptor` (page 114)
Creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

`CreateLogicalDescriptor` (page 115)
Creates a logical descriptor that specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

`CreateObjSpecifier` (page 116)
Assembles an object specifier that identifies one or more Apple event objects, from other descriptors.

`CreateOffsetDescriptor` (page 117)
Creates an offset descriptor that specifies the position of an element in relation to the beginning or end of its container.

`CreateRangeDescriptor` (page 118)
Creates a range descriptor that specifies a series of consecutive elements in the same container.

Deallocating Memory for Descriptors

`AEDisposeDesc` (page 40)
Deallocates the memory used by a descriptor.

Deallocating Memory for Tokens

`AEDisposeToken` (page 41)
Deallocates the memory used by a token.

Deleting Descriptors

`AEDeleteItem` (page 38)
Deletes a descriptor from a descriptor list, causing all subsequent descriptors to move up one place.

`AEDeleteKeyDesc` (page 39)
Deletes a keyword-specified parameter from an Apple event record.

`AEDeleteParam` (page 39)
Deletes a keyword-specified parameter from an Apple event record.

Dispatching Apple Events

`AEProcessAppleEvent` (page 73)
Calls the handler, if one exists, for a specified Apple event.

Getting, Calling, and Removing Object Accessor Functions

`AECallObjectAccessor` (page 28)
Invokes the appropriate object accessor function for a specific desired type and container type.

`AEGetObjectAccessor` (page 57)
Gets an object accessor function from an object accessor dispatch table.

`AEInstallObjectAccessor` (page 67)
Adds or replaces an entry for an object accessor function to an object accessor dispatch table.

`AERemoveObjectAccessor` (page 86)
Removes an object accessor function from an object accessor dispatch table.

Getting Data or Descriptors From Apple Events and Apple Event Records

`AEGetAttributeDesc` (page 45)
Gets a copy of the descriptor for a specified Apple event attribute from an Apple event; typically used when your application needs to pass the descriptor on to another function.

`AEGetAttributePtr` (page 46)
Gets a copy of the data for a specified Apple event attribute from an Apple event; typically used when your application needs to work with the data directly.

`AEGetKeyDesc` (page 52)
Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event record

`AEGetKeyPtr` (page 53)
Gets a copy of the data for a specified Apple event parameter from an Apple event record.

`AEGetParamDesc` (page 59)
Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event or an Apple event record.

`AEGetParamPtr` (page 60)
Gets a copy of the data for a specified Apple event parameter from an Apple event or an Apple event record.

Getting Information About the Apple Event Manager

`AEManagerInfo` (page 70)
Provides information about the version of the Apple Event Manager currently available or the number of processes that are currently recording Apple events.

Getting Items From Descriptor Lists

`AEGetArray` (page 44)
Extracts data from an Apple event array created with the `AEPutArray` function and stores it as a standard array of fixed size items in the specified buffer.

`AEGetNthDesc` (page 55)
Copies a descriptor from a specified position in a descriptor list into a specified descriptor; typically used when your application needs to pass the extracted data to another function as a descriptor.

`AEGetNthPtr` (page 56)
Gets a copy of the data from a descriptor at a specified position in a descriptor list; typically used when your application needs to work with the extracted data directly.

Getting the Sizes and Descriptor Types of Descriptors

`AESizeOfAttribute` (page 98)
Gets the size and descriptor type of an Apple event attribute from a descriptor of type `AppleEvent`.

`AESizeOfKeyDesc` (page 99)
Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord`.

`AESizeOfNthItem` (page 100)
Gets the data size and descriptor type of the descriptor at a specified position in a descriptor list.

`AESizeOfParam` (page 101)
Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord` or `AppleEvent`.

Initializing the Object Support Library

`AEObjectInit` (page 71)
Initializes the Object Support Library.

`AESetObjectCallbacks` (page 96)
Specifies the object callback functions for your application.

Locating Processes on Remote Computers

Available starting in Mac OS X version v10.3, these functions allow you to locate processes on remote computers (a task supported by the PPCToolbox in Mac OS 9).

`AECreateRemoteProcessResolver` (page 36)
Creates an object for resolving a list of remote processes.

`AEDisposeRemoteProcessResolver` (page 40)
Disposes of an `AERemoteProcessResolverRef`.

`AERemoteProcessResolverGetProcesses` (page 82)
Returns an array of objects containing information about processes running on a remote machine.

`AERemoteProcessResolverScheduleWithRunLoop` (page 83)
Schedules a resolver for execution on a given run loop in a given mode.

Managing Apple Event Dispatch Tables

`AEGetEventHandler` (page 51)
Gets an event handler from an Apple event dispatch table.

`AEInstallEventHandler` (page 65)
Adds an entry for an event handler to an Apple event dispatch table.

`AERemoveEventHandler` (page 85)
Removes an event handler entry from an Apple event dispatch table.

Managing Coercion Handler Dispatch Tables

`AEGetCoercionHandler` (page 47)
Gets the coercion handler for a specified descriptor type.

`AEInstallCoercionHandler` (page 64)
Installs a coercion handler in either the application or system coercion handler dispatch table.

`AERemoveCoercionHandler` (page 84)
Removes a coercion handler from a coercion handler dispatch table.

Managing Special Handler Dispatch Tables

`AEGetSpecialHandler` (page 62)
Gets a specified handler from a special handler dispatch table.

`AEInstallSpecialHandler` (page 68)
Installs a callback function in a special handler dispatch table.

`AERemoveSpecialHandler` (page 87)
Removes a handler from a special handler dispatch table.

Operating On Descriptor Data

`AEGetDescData` (page 48)
Gets the data from the specified descriptor.

`AEGetDescDataSize` (page 50)
Gets the size, in bytes, of the data in the specified descriptor.

`AEGetDescDataRange` (page 49)
Retrieves a specified series of bytes from the specified descriptor.

`AEReplaceDescData` (page 88)
Copies the specified data into the specified descriptor, replacing any previous data.

Requesting More Time to Respond to Apple Events

`AEResetTimer` (page 88)
Resets the timeout value for an Apple event to its starting value.

Requesting User Interaction

`AEGetInteractionAllowed` (page 52)
Gets your application's current user interaction preferences for responding to an Apple event as a server application.

`AEInteractWithUser` (page 69)
Initiates interaction with the user when your application is a server application responding to an Apple event.

`AESetInteractionAllowed` (page 95)
Specifies user interaction preferences for responding to an Apple event when your application is the server application.

Resolving Object Specifiers

`AEResolve` (page 89)
Resolves an object specifier.

Sending an Apple Event

`AESend` (page 92)
Sends the specified Apple event.

Creating Apple Event Structures in Memory

`AEBuildAppleEvent` (page 24)
Constructs an entire Apple event in a single call.

`AEBuildDesc` (page 26)
Provides a facility for compiling AEBuild descriptor strings into Apple event descriptors (`AEDesc`).

`AEBuildParameters` (page 27)
Adds additional parameters or attributes to an existing Apple event.

`AEPrintDescToHandle` (page 72)
Provides a pretty printer facility for displaying the contents of Apple event descriptors.

`vAEBuildAppleEvent` (page 135)
Allows you to encapsulate calls to `AEBuildAppleEvent` in a wrapper routine.

`vAEBuildDesc` (page 136)
Allows you to encapsulate calls to `AEBuildDesc` in your own wrapper routines.

`vAEBuildParameters` (page 137)
Allows you to encapsulate calls to `AEBuildParameters` in your own `stdarg`-style wrapper routines, using techniques similar to those allowed by vsprintf.

Creating Apple Event Structures Using Streams

`AEStreamClose` (page 101)
Closes and deallocates an `AEStreamRef`.

`AEStreamCloseDesc` (page 102)
Marks the end of a descriptor in an `AEStreamRef`.

`AEStreamCloseList` (page 102)
Marks the end of a list of descriptors in an `AEStreamRef`.

`AEStreamCloseRecord` (page 103)
Marks the end of a record in an `AEStreamRef`.

`AEStreamCreateEvent` (page 103)
Creates a new Apple event and opens a stream for writing data to it.

`AEStreamOpen` (page 105)
Opens a new `AEStreamRef` for use in building a descriptor.

`AEStreamOpenDesc` (page 105)
Marks the beginning of a descriptor in an `AEStreamRef`.

`AEStreamOpenEvent` (page 106)
Opens a stream for an existing Apple event.

`AEStreamOpenKeyDesc` (page 106)
Marks the beginning of a key descriptor in an `AEStreamRef`.

`AEStreamOpenList` (page 107)
Marks the beginning of a descriptor list in an `AEStreamRef`.

`AEStreamOpenRecord` (page 107)
Marks the beginning of an Apple event record in an `AEStreamRef`.

`AEStreamOptionalParam` (page 108)
Designates a parameter in an Apple event as optional.

`AEStreamSetRecordType` (page 109)
Sets the type of the most recently created record in an `AEStreamRef`.

`AEStreamWriteAEDesc` (page 109)
Copies an existing descriptor into an `AEStreamRef`.

`AEStreamWriteData` (page 110)
Appends data to the current descriptor in an `AEStreamRef`.

`AEStreamWriteDesc` (page 110)
Appends the data for a complete descriptor to an `AEStreamRef`.

`AEStreamWriteKey` (page 111)
Marks the beginning of a keyword/descriptor pair for a descriptor in an `AEStreamRef`.

`AEStreamWriteKeyDesc` (page 112)
Writes a complete keyword/descriptor pair to an `AEStreamRef`.

Working With Lower Level Apple Event Functions

`AEGetRegisteredMachPort` (page 61)
Returns the Mach port (in the form of a `mach_port_t`) that was registered with the bootstrap server for this process.

`AEDecodeMessage` (page 37)
Decodes a Mach message and converts it into an Apple event and its related reply.

`AESendMessage` (page 94)
Sends an AppleEvent to a target process without some of the overhead required by `AESend`.

`AEProcessMessage` (page 74)
Decodes and dispatches a low level Mach message event to an event handler, including packaging and returning the reply to the sender.

Serializing Apple Event Data

`AESizeOfFlattenedDesc` (page 99)
Returns the amount of buffer space needed to store the descriptor after flattening it.

`AEFlattenDesc` (page 42)
Flattens the specified descriptor and stores the data in the supplied buffer.

`AEUnflattenDesc` (page 114)
Unflattens the data in the passed buffer and creates a descriptor from it.

Suspending and Resuming Apple Event Handling

`AEGetTheCurrentEvent` (page 63)
Gets the Apple event that is currently being handled.

`AEResumeTheCurrentEvent` (page 90)
Informs the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event.

`AESetTheCurrentEvent` (page 97)
: Specifies a current Apple event to take the place of the one your application has suspended.

`AESuspendTheCurrentEvent` (page 113)
: Suspends the processing of the Apple event that is currently being handled.

Miscellaneous

`AECheckIsRecord` (page 29)
: Determines whether a descriptor is truly an `AERecord`.

`AEInitializeDesc` (page 64)
: Initializes a new descriptor.

Functions

AEBuildAppleEvent

Constructs an entire Apple event in a single call.

```
OSStatus AEBuildAppleEvent (
   AEEventClass theClass,
   AEEventID theID,
   DescType addressType,
   const void *addressData,
   Size addressLength,
   SInt16 returnID,
   SInt32 transactionID,
   AppleEvent *result,
   AEBuildError *error,
   const char *paramsFmt,
   ...
);
```

Parameters

theClass
: The event class for the resulting Apple event. See `AEEventClass` (page 171).

theID
: The event id for the resulting Apple event. See `AEEventID` (page 172).

addressType
: The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`. See `DescType` (page 176).

addressData
: A pointer to the address information.

addressLength
: The number of bytes pointed to by the `addressData` parameter.

returnID

The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID.

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events.

result

A pointer to a descriptor where the resulting descriptor should be stored. See `AppleEvent` (page 175) for a description of the data type.

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 162) for a description of the data type.

paramsFmt

An AEBuild format string describing the AppleEvent record to be created. The format of these strings is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends. That technote also describes possible error return codes for syntax errors in the format string.

Return Value

A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the *error* parameter to discover information about other errors. See "Apple Event Manager Result Codes" (page 252).

Discussion

IMPORTANT: Following the parameters described above, the `AEBuildAppleEvent` function takes a variable number of parameters as specified by the format string provided in the *paramsFmt* parameter.

This function and related "AEBuild" routines (including `AEBuildDesc` (page 26) and `AEBuildParameters` (page 27), and the variable-argument versions, `vAEBuildAppleEvent` (page 135), `vAEBuildDesc` (page 136), and `vAEBuildParameters` (page 137)) provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildAppleEvent` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

In many ways, the AEBuild routines are very much like the standard C library's `printf` suite of routines. The syntax for the format string that you provide is very simple and allows for the substitution of data items into the Apple event descriptors being created.

The `AEBuildAppleEvent` function is similar to `AECreateAppleEvent` (page 32), but in addition to creating the Apple event, it also constructs the parameters for the event from the last three arguments. You can use `AEBuildAppleEvent` to build an entire Apple event, or `AEBuildParameters` (page 27) to add additional parameters to an existing Apple event.

The syntax of the formatting string for an entire Apple event (as passed to `AEBuildAppleEvent`) is almost identical to that used to represent the contents of an Apple event, without the curly braces. The event is defined as a sequence of name-value pairs, with optional parameters preceded with a tilde (~) character. The syntax is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends.

It is important to note that the identifier for the direct parameter in an Apple event, specified by the constant `keyDirectObject`, is four minus signs (`'----'`). The minus sign has special meaning in AEBuild strings, and it should always be enclosed in single quotes when it is used to identify the direct parameter for an Apple event in a descriptor string.

Version Notes
Prior to Mac OS X version 10.3, `AEBuildAppleEvent` would fail if you supplied a data parameter with size greater than 32767 bytes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEBuildDesc

Provides a facility for compiling AEBuild descriptor strings into Apple event descriptors (`AEDesc`).

```
OSStatus AEBuildDesc (
   AEDesc *dst,
   AEBuildError *error,
   const char *src,
   ...
);
```

Parameters

dst

A pointer to a descriptor where the resulting descriptor should be stored. See `AEDesc` (page 162).

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 162).

src

An *`AEBuild`* format string describing the descriptor to be created.

Return Value
A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the *`error`* parameter to discover information about other errors. See also "Apple Event Manager Result Codes" (page 252).

Discussion
This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

For additional information on using the AEBuild routines, see the descriptions for `AEBuildAppleEvent` (page 24) and `AEBuildParameters` (page 27).

Version Notes
Prior to Mac OS X version 10.3, `AEBuildDesc` would fail if you supplied a data parameter with size greater than 32767 bytes.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEBuildParameters

Adds additional parameters or attributes to an existing Apple event.

```
OSStatus AEBuildParameters (
   AppleEvent *event,
   AEBuildError *error,
   const char *format,
   ...
);
```

Parameters

event
The Apple event to which you are adding parameters. See `AppleEvent` (page 175).

error
A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 162).

format
An *AEBuild* format string describing the parameters to be created.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function can be called more than once to add any desired number of parameters or attributes to an existing Apple event. The Apple event should already have been created through either a call to `AECreateAppleEvent` (page 32) or `AEBuildAppleEvent` (page 24).

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `AEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

For additional information on using the AEBuild routines, see the descriptions for `AEBuildAppleEvent` (page 24) and `AEBuildDesc` (page 26).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AECallObjectAccessor

Invokes the appropriate object accessor function for a specific desired type and container type.

```
OSErr AECallObjectAccessor (
   DescType desiredClass,
   const AEDesc *containerToken,
   DescType containerClass,
   DescType keyForm,
   const AEDesc *keyData,
   AEDesc *token
);
```

Parameters

desiredClass

The type of the Apple event object requested. Some possible values are defined in "Object Class ID Constants" (page 215). See `DescType` (page 176).

containerToken

A pointer to the token that identifies the container for the desired object. (Token is defined in `AEDisposeToken` (page 41).) See `AEDesc` (page 162).

containerClass

The object class of the container for the desired objects. See `DescType` (page 176).

keyForm

The key form that specifies how to find the object within the container. Key form constants are described in "Key Form and Descriptor Type Object Specifier Constants" (page 206). See `DescType` (page 176).

keyData

A pointer to the key data that identifies the object within the container. The type of this data is form-specific. That is, `formName` typically has key data of type `typeText`. See `AEDesc` (page 162).

token

A pointer to a token. On return, a token specifying the desired object (or objects). Your application should dispose of this token when it is through with it by calling `AEDisposeToken` (page 41). See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). `AECallObjectAccessor` returns any result codes returned by the object accessor function it calls.

Discussion

If you want your application to do some of the Apple event object resolution normally performed by the `AEResolve` (page 89) function, you can use `AECallObjectAccessor` to invoke an object accessor function. This might be useful, for example, if you have installed an object accessor function using `typeWildCard` for the `AEInstallObjectAccessor` function's `desiredClass` parameter and `typeAEList` for the `containerType` parameter. To return a list of tokens for a request like "line one of every window" the object accessor function can create an empty list, then call `AECallObjectAccessor` for each requested element, adding tokens for each element to the list one at a time.

The parameters of `AECallObjectAccessor` are identical to the parameters of an object accessor function, as described in `OSLAccessorProcPtr` (page 149) with one exception—the Apple Event Manager adds a reference constant parameter each time it calls the object accessor function.

You can also call a specific object accessor function directly through its universal procedure pointer with one of the invoke functions described in "Creating, Calling, and Deleting Universal Procedure Pointers" (page 15).

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AECheckIsRecord

Determines whether a descriptor is truly an `AERecord`.

```
Boolean AECheckIsRecord (
   const AEDesc *theDesc
);
```

Parameters

`theDesc`
A pointer to the descriptor to check.

Return Value
Returns `true` if the descriptor is an `AERecord` or an `AppleEvent`, `false` otherwise.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoerceDesc

Coerces the data in a descriptor to another descriptor type and creates a descriptor containing the newly coerced data.

```
OSErr AECoerceDesc (
   const AEDesc *theAEDesc,
   DescType toType,
   AEDesc *result
);
```

Parameters

`theAEDesc`
A pointer to the descriptor containing the data to coerce. See `AEDesc` (page 162).

toType

The desired descriptor type of the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

result

A pointer to a descriptor. On successful return, a descriptor containing the coerced data and matching the descriptor type specified in `toType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). If `AECoerceDesc` returns a nonzero result code, it returns a null descriptor record (a descriptor record of type `typeNull`, which does not contain any data) unless the Apple Event Manager is not available because of limited memory.

Version Notes

See the Version Notes section for the `AECoercePtr` (page 30) function for information on when to use descriptor-based versus pointer-based coercion handlers starting in Mac OS X version 10.2.

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

Declared In

`AEDataModel.h`

AECoercePtr

Coerces data to a desired descriptor type and creates a descriptor containing the newly coerced data.

```
OSErr AECoercePtr (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   DescType toType,
   AEDesc *result
);
```

Parameters

typeCode

The descriptor type of the source data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to the data to coerce.

dataSize

The length, in bytes, of the data to coerce.

toType

The desired descriptor type of the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

result

A pointer to a descriptor. On successful return, a descriptor containing the coerced data and matching the descriptor type specified in `toType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Starting in Mac OS X version 10.2, pointer-based coercion handlers are not called if the input type is "structured"—that is, if the type to be coerced is `typeAEList`, `typeAERecord`, or coerced `typeAERecord`. If you want to add a coercion handler for one of these types, it must be a descriptor-based handler. This does not mean you are required to use descriptor-based coercion handlers everywhere—for "flat" data types, such as `typeText`, pointer-based handlers are still fine.

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECountItems

Counts the number of descriptors in a descriptor list.

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

Parameters

theAEDescList

A pointer to the descriptor list to count. See `AEDescList` (page 169).

theCount

A pointer to a count variable. On return, the number of descriptors in the specified descriptor list, which can be 0, if the list is empty.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Your application typically counts the descriptors in a descriptor list when it is extracting data from an Apple event. You can use the functions in "Getting Items From Descriptor Lists" to get an individual item from a descriptor list or to iterate through the items.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`AEDataModel.h`

AECreateAppleEvent

Creates an Apple event with several important attributes but no parameters.

```
OSErr AECreateAppleEvent (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   const AEAddressDesc *target,
   AEReturnID returnID,
   AETransactionID transactionID,
   AppleEvent *result
);
```

Parameters

theAEEventClass

The event class of the Apple event to create. This parameter becomes accessible through the `keyEventClassAttr` attribute of the Apple event. Some event classes are described in "Event Class Constants" (page 201). See `AEEventClass` (page 171).

theAEEventID

The event ID of the Apple event to create. This parameter becomes accessible through the `keyEventIDAttr` attribute of the Apple event. Some event IDs are described in "Event ID Constants" (page 202). See `AEEventID` (page 172).

target

A pointer to an address descriptor. Before calling `AECreateAppleEvent`, you set the descriptor to identify the target (or server) application for the Apple event. This parameter becomes accessible through the `keyAddressAttr` attribute of the Apple event. See `AEAddressDesc` (page 167).

returnID

The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID. This parameter becomes accessible through the `keyReturnIDAttr` attribute of the Apple event. The return ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 205). See `AEReturnID` (page 174).

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events. This parameter becomes accessible through the `keyTransactionIDAttr` attribute of the Apple event. This transaction ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 205). See `AETransactionID` (page 175).

result

A pointer to an Apple event. On successful return, the new Apple event. On error, a null descriptor (one with descriptor type `typeNull`). If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting Apple event after it has finished using it. See the `AppleEvent` (page 175) data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AECreateAppleEvent` function creates an empty Apple event. You can add parameters to the Apple event after you create it with the functions described in "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 14).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateDesc

Creates a new descriptor that incorporates the specified data.

```
OSErr AECreateDesc (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   AEDesc *result
);
```

Parameters

typeCode

The descriptor type for the new descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to the data for the new descriptor. This data is copied into a newly-allocated block of memory for the descriptor that is created. To minimize copying overhead, consider using `AECreateDescFromExternalPtr` (page 34).

dataSize

The length, in bytes, of the data for the new descriptor.

result

A pointer to a descriptor. On successful return, a descriptor that incorporates the data specified by the `dataPtr` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
While it is possible to create an Apple event descriptor or a descriptor list or a descriptor with the `AECreateDesc` function (assuming you have access to the raw data for an Apple event, list, or descriptor), you typically create these structured objects with their specific creation routines—`AECreateAppleEvent`, `AECreateList`, or `AECreateDesc`.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateDescFromExternalPtr

Creates a new descriptor that uses a memory buffer supplied by the caller.

```
OSStatus AECreateDescFromExternalPtr (
   OSType descriptorType,
   const void *dataPtr,
   Size dataLength,
   AEDisposeExternalUPP disposeCallback,
   SRefCon disposeRefcon,
   AEDesc *theDesc
);
```

Parameters

descriptorType
The descriptor type for the new descriptor.

dataPtr
A pointer to the data for the new descriptor. The memory that is pointed to cannot be a `Handle` (which may move in memory), cannot be modified by the caller, and must be preserved in place (and not freed), until the *disposeCallback* function is called.

If possible, the descriptor will be mapped into the address space of the recipient using shared memory, avoiding an actual memory copy.

The pointer that is passed in does not need to be aligned to any particular boundary, but is optimized to transfer data on a page boundary. You can get the current page size (4096 on all current Mac OS X systems) with the `getpagesize(3)` call. (Type `man 3 getpagesize` in a Terminal window for documentation.)

dataLength
The length, in bytes, of the data for the new descriptor.

disposeCallback
A universal procedure pointer to a dispose callback function of type `AEDisposeExternalProcPtr` (page 143). Your callback function will be called when the block of memory provided by `dataPtr` is no longer needed by the Apple Event Manager. The function can be called at any time, including during creation of the descriptor.

disposeRefcon

A reference constant the Apple Event Manager passes to the `disposeCallback` function whenever it calls the function. If your dispose function doesn't require a reference constant, pass 0 for this parameter.

theDesc

A pointer to a descriptor. On successful return, a descriptor that incorporates the data specified by the `dataPtr` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function is different than `AECreateDesc` (page 33), in that it creates a descriptor that uses the data block provided by the caller "in place," rather than allocate a block of memory and copy the data to it. This function can provide dramatically improved performance if you're working with large chunks of data. It attempts to copy the descriptor to the address space of any recipient process using virtual memory APIs, avoiding an actual memory copy. For example, you might want to use this function to pass a large image in an Apple event.

You can use the `AEGetDescDataRange` (page 49) function to access a specific section of a large block of data.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AECreateList

Creates an empty descriptor list or Apple event record.

```
OSErr AECreateList (
   const void *factoringPtr,
   Size factoredSize,
   Boolean isRecord,
   AEDescList *resultList
);
```

Parameters

factoringPtr

A pointer to the data at the beginning of each descriptor that is the same for all descriptors in the list. If there is no common data, or if you decide not to isolate the common data, pass `NULL` as the value of this parameter.

factoredSize

The size of the common data. If there is no common data, or if you decide not to isolate the common data, pass 0 as the value of this parameter. (See the Discussion section for more information.)

isRecord

A Boolean value that specifies the kind of list to create. Pass a value of `TRUE` to create an Apple event record (a data structure of type `AERecord` (page 173)) or `FALSE` to create a descriptor list.

resultList

A pointer to a descriptor list variable. On successful return, the descriptor list or Apple event record that the `AECreateList` function creates. On error, a null descriptor. See `AEDescList` (page 169).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AECreateList` function creates an empty descriptor list or Apple event record. You can use the functions described in "Adding Items to Descriptor Lists" to populate the list as part of creating an Apple event. After sending the Apple event with the `AESend` (page 92) function, you should dispose of the descriptor list with the `AEDisposeDesc` (page 40) function when you no longer need it.

If you intend to use a descriptor list for a factored Apple event array, you must provide, in the `factoringPtr` parameter, a pointer to the data shared by all items in the array and, in the `factoredSize` parameter, the size of the common data. The common data must be 4, 8, or more than 8 bytes in length because it always consists of (a) the descriptor type (4 bytes) (b) the descriptor type (4 bytes) and the size of each item's data (4 bytes) or (c) the descriptor type (4 bytes), the size of each item's data (4 bytes), and some portion of the data itself (1 or more bytes).

For information about data types used with Apple event arrays, see "Apple Event Manager Data Types" (page 161).

Version Notes
The *factoringPtr* and *factoredSize* parameters are not supported in Mac OS X v10.2 and later. You should pass `NULL` and zero, respectively, for these parameters.

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECreateRemoteProcessResolver

Creates an object for resolving a list of remote processes.

```
AERemoteProcessResolverRef AECreateRemoteProcessResolver (
   CFAllocatorRef allocator,
   CFURLRef url
);
```

Parameters

allocator

An object that is used to allocates and deallocate any Core Foundation types created or returned by this API. You can pass `kCFAllocatorDefault` to get the default allocation behavior. The allocator is based on `CFAllocatorRef`, an opaque data type described in the Core Foundation Reference Documentation.

url

A CFURL reference identifying the remote host and port on which to look for processes. See the Core Foundation Reference Documentation for a description of the CFURLRef data type.

Return Value
An AERemoteProcessResolverRef (page 173), which must be disposed of with AEDisposeRemoteProcessResolver (page 40). A resolver can only be used one time; once it has obtained a list of remote processes from a server, or gotten an error, it can no longer be scheduled. To retrieve a new list of processes, create a new instance of this object.

Discussion
You supply this function with the URL for a remote host and port; it returns a reference to a resolver object. To obtain a list of remote processes from the resolver, you can query it synchronously with AERemoteProcessResolverGetProcesses (page 82), which blocks until the request completes (either successfully or with an error).

If asynchronous behavior is desired, you can optionally use AERemoteProcessResolverScheduleWithRunLoop (page 83) to schedule the resolver asynchronously on a run loop. If so, you supply a callback routine (see AERemoteProcessResolverCallback (page 148)) that is executed when the resolver completes. To obtain information about the remote processes, you will again have to call AERemoteProcessResolverGetProcesses (page 82).

A resolver can only be used once; once it has fetched the data or gotten an error it can no longer be scheduled. The data obtained by the resolver is a CFArrayRef of CFDictionaryRef objects. For information on the format of the returned remote process information, see the description of the function result for the function AERemoteProcessResolverGetProcesses (page 82), and also "Remote Process Dictionary Keys" (page 218).

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
AppleEvents.h

AEDecodeMessage

Decodes a Mach message and converts it into an Apple event and its related reply.

```
OSStatus AEDecodeMessage (
   mach_msg_header_t *header,
   AppleEvent *event,
   AppleEvent *reply
);
```

Parameters

header

A pointer to a Mach message header for the event to be decoded.

event

A pointer to a null Apple event descriptor (one with descriptor type `typeNull`). On successful completion, contains the decoded Apple event. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it.

reply

A pointer to a null Apple event descriptor. On successful completion, contains the reply event from the decoded Apple event. To send the reply, you use the following:

```
AESendMessage(reply, NULL, kAENoReply, kAEDefaultTimeout);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The Apple Event Manager provides the following functions (on Mac OS X only) for working with Apple events at a lower level: `AEGetRegisteredMachPort` (page 61), `AEDecodeMessage`, `AESendMessage` (page 94), and `AEProcessMessage` (page 74). See the descriptions for those functions for more information on when you might use them.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AEDeleteItem

Deletes a descriptor from a descriptor list, causing all subsequent descriptors to move up one place.

```
OSErr AEDeleteItem (
   AEDescList *theAEDescList,
   long index
);
```

Parameters

theAEDescList

A pointer to the descriptor list containing the descriptor to delete. See `AEDescList` (page 169).

index

A one-based positive integer indicating the position of the descriptor to delete. `AEDeleteItem` returns an error if you pass zero, a negative number, or a value that is out of range.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDeleteKeyDesc

Deletes a keyword-specified parameter from an Apple event record.

```
OSErr AEDeleteKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword
);
```

Parameters

theAERecord

A pointer to the Apple event record to delete the parameter from.

theAEKeyword

The keyword that specifies the parameter to delete. Some keyword constants are described in "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

This function is declared as a macro that invokes `AEDeleteParam` (page 39), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes

See `AEDeleteParam` (page 39).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEDeleteParam

Deletes a keyword-specified parameter from an Apple event record.

```
OSErr AEDeleteParam (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword
);
```

Parameters

theAppleEvent

A pointer to the Apple event or Apple event record to delete the parameter from. See `AppleEvent` (page 175).

theAEKeyword

The keyword that specifies the parameter to delete. Some keyword constants are described in "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEDisposeDesc

Deallocates the memory used by a descriptor.

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

Parameters

theAEDesc

A pointer to the descriptor to deallocate. On return, a null descriptor. If you pass a null descriptor in this parameter, AEDisposeDesc returns noErr. See AEDesc (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). As currently implemented, AEDisposeDesc always returns noErr.

Discussion
The AEDisposeDesc function deallocates the memory used by a descriptor. After calling this method, the descriptor becomes an empty descriptor with a type of typeNULL. Because all Apple event structures (except for keyword-specified descriptors) are descriptors, you can use AEDisposeDesc for any of them.

Do not call AEDisposeDesc on a descriptor obtained from another Apple Event Manager function (such as the reply event from a call to AESend (page 92)) unless that function returns successfully.

Special Considerations

If the AEDesc might contain an OSL token, dispose of it with AEDisposeToken (page 41).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
AEDataModel.h

AEDisposeRemoteProcessResolver

Disposes of an AERemoteProcessResolverRef.

```
void AEDisposeRemoteProcessResolver (
   AERemoteProcessResolverRef ref
);
```

Parameters

ref

The `AERemoteProcessResolverRef` (page 173) to dispose of. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 36).

Discussion
If this resolver is currently scheduled on a run loop, it is unscheduled, and the asynchronous callback is not executed.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AEDisposeToken

Deallocates the memory used by a token.

```
OSErr AEDisposeToken (
   AEDesc *theToken
);
```

Parameters

theToken

A pointer to the token to dispose of. On successful return, the pointer is set to the null descriptor. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Your application calls the `AEResolve` (page 89) function to resolve an object specifier, with the help of the object accessor functions described in "Object Accessor Callbacks" and the application object callback functions described in "Object Callback Functions".

When `AEResolve` returns a final token to your event handler as the result of the resolution of an object specifier, your application must deallocate the memory used by the token. When your application calls the `AEDisposeToken` function, the Apple Event Manager first calls your application's token disposal function, if you have provided one. The token disposal function is described in `OSLDisposeTokenProcPtr` (page 155).

If you haven't provided a token disposal function, or if your application's token disposal function returns `errAEEventNotHandled` as the function result, the Apple Event Manager calls the system token disposal function if one is available. If there is no system token disposal function or the function returns `errAEEventNotHandled` as the function result, the Apple Event Manager calls the `AEDisposeDesc` function to dispose of the token.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEDuplicateDesc

Creates a copy of a descriptor.

```
OSErr AEDuplicateDesc (
   const AEDesc *theAEDesc,
   AEDesc *result
);
```

Parameters

theAEDesc

A pointer to the descriptor to duplicate. See `AEDesc` (page 162).

result

A pointer to a descriptor. On return, the descriptor contains a copy of the descriptor specified by the `theAEDesc` parameter. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
It is common for applications to send Apple events that have one or more attributes or parameters in common. For example, if you send a series of Apple events to the same application, the address attribute is the same. In these cases, the most efficient way to create the necessary Apple events is to make a template Apple event that you can then copy—by calling the `AEDuplicateDesc` function—as needed. You then fill in or change the remaining parameters and attributes of the copy, send the copy by calling the `AESend` (page 92) function and, after `AESend` returns a result code, dispose of the copy by calling `AEDisposeDesc` (page 40). You can use this approach to prepare structures of type `AEDesc` (page 162), `AEDescList` (page 169), `AERecord` (page 173), and `AppleEvent` (page 175).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFlattenDesc

Flattens the specified descriptor and stores the data in the supplied buffer.

```
OSStatus AEFlattenDesc (
   const AEDesc *theAEDesc,
   Ptr buffer,
   Size bufferSize,
   Size *actualSize
);
```

Parameters

theAEDesc

A pointer to the descriptor to be flattened. See `AEDesc` (page 162).

buffer

A pointer to memory, allocated by the application, where the flattened data will be stored. See the *bufferSize* parameter for information on how large a buffer you should allocate.

bufferSize

The size of the buffer pointed to by buffer. Prior to calling `AEFlattenDesc`, you call the `AESizeOfFlattenedDesc` (page 99) function to determine the required size of the buffer for the flatten operation.

If *bufferSize* is too small, `AEFlattenDesc` returns `errAEBufferTooSmall` and doesn't store any data in the buffer.

actualSize

A pointer to a size variable. On return, the variable contains the actual size of the flattened data. You can specify `NULL` for this parameter if you do not care about the returned size.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

Flattening a descriptor serializes the data it contains. That is, it reduces a complex, possibly deeply nested structure to a series of bytes that can conveniently be stored. The descriptor can be reconstituted from the stored bytes with the `AEUnflattenDesc` (page 114) function.

Applications can be scriptable and work with Apple events without needing to flatten and unflatten descriptors. Flattening is a special-purpose capability that is useful in circumstances where it may be convenient to store data by saving and restoring a descriptor, rather than having to manually extract the data from it, store the data as a separate step, then manually recreate the descriptor (if necessary). For example, you might use flattening to store a preference setting received through an Apple event.

Flattening and unflattening should work without loss of data on descriptors that represent `AEDesc`, `AEList`, and `AERecord` structures. You can also use the process with `AppleEvent` descriptors. However, keep in mind that Apple events may contain attributes that are relevant only to a running process, and these attributes may not keep their meaning when the event is reconstituted.

Flattening and unflattening works across OS versions, including between Mac OS 9 and Mac OS X.

Flattening is endian-neutral. That is, you can save flattened data on a machine that is either big-endian or little-endian, then retrieve and unflatten the data on either type of machine, without any special steps by your application.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEGetArray

Extracts data from an Apple event array created with the AEPutArray function and stores it as a standard array of fixed size items in the specified buffer.

```
OSErr AEGetArray (
   const AEDescList *theAEDescList,
   AEArrayType arrayType,
   AEArrayDataPointer arrayPtr,
   Size maximumSize,
   DescType *itemType,
   Size *itemSize,
   long *itemCount
);
```

Parameters

theAEDescList

A pointer to the descriptor list to get the array from. If the array is of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor list must be factored. A factored descriptor list is one in which the Apple Event Manager automatically isolates the data that is common to all the elements of the list so that the common data only appears in the list once. To create a factored descriptor list, you call the AECreateList (page 35) function and specify the data that is common to all elements in the descriptor array. See the Discussion section for related information. See AEDescList (page 169).

arrayType

The Apple event array type to convert. Pass one of the constants: described in "Data Array Constants" (page 196). See AEArrayType (page 168).

arrayPtr

A pointer to a buffer, allocated and disposed of by your application, for storing the array. The size in bytes must be at least as large as the value you pass in the maximumSize parameter. On return, the buffer contains the array of fixed-size items. See AEArrayDataPointer (page 167).

maximumSize

The maximum length, in bytes, of the expected data. The AEGetArray function will not return more data than you specify in this parameter.

itemType

A pointer to a descriptor type. On return, for arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor type of the items in the returned array. The AEGetArray function doesn't supply a value in itemType for arrays of type kAEDescArray and kAEKeyDescArray because they may contain descriptors of different types. Possible descriptor types are listed in "Descriptor Type Constants" (page 197). See DescType (page 176).

itemSize

A pointer to a size variable. On return, for arrays of type kAEDataArray or kAEPackedArray, the size (in bytes) of each item in the returned array. You don't get an item size for arrays of type kAEDescArray, kAEKeyDescArray, or kAEHandleArray because descriptors and handles (though not the data they point to) have a known size.

itemCount

A pointer to a size variable. On return, the number of items in the returned array.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AEGetArray` function uses a buffer identified by the pointer in the `arrayPtr` parameter to store the converted data for the Apple event array specified by the `theAEDescList` parameter. For example, `AEGetArray` may convert an array of descriptors of type `typeLongInteger` into a simple array of integer values or an array of descriptors of type `typeFSS` into an array of file specification records.

Even if the descriptor list that contains the array is factored, the converted data for each array item includes the data common to all the descriptors in the list. The Apple Event Manager automatically reconstructs the common data for each item when you call `AEGetArray`.

For information about creating and factoring descriptor lists for Apple event arrays, see `AECreateList` (page 35). For information about adding an Apple event array to a descriptor list, see `AEPutArray` (page 75).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetAttributeDesc

Gets a copy of the descriptor for a specified Apple event attribute from an Apple event; typically used when your application needs to pass the descriptor on to another function.

```
OSErr AEGetAttributeDesc (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   AEDesc *result
);
```

Parameters

`theAppleEvent`
A pointer to the Apple event to get the attribute descriptor from. See `AppleEvent` (page 175).

`theAEKeyword`
The keyword that specifies the desired attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 209). See `AEKeyword` (page 172).

`result`
A pointer to a descriptor. On successful return, a copy of the specified Apple event attribute, coerced, if necessary, to the descriptor type specified in `desiredType`. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
To get Apple event attribute data for your application to use directly, call `AEGetAttributePtr` (page 46). To get a descriptor for an Apple event attribute to pass on to another Apple Event Manager routine, call `AEGetAttributeDesc`.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetAttributePtr

Gets a copy of the data for a specified Apple event attribute from an Apple event; typically used when your application needs to work with the data directly.

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

Parameters

theAppleEvent

A pointer to the Apple event to get the attribute data from. See `AppleEvent` (page 175).

theAEKeyword

The keyword that specifies the desired attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 209). See `AEKeyword` (page 172).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetAttributePtr` attempts to coerce the data to this type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned data is the same as the descriptor type of the Apple event attribute.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

See `DescType` (page 176).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the attribute data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to a buffer, local variable, or other storage location, created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the attribute data.

maximumSize

The maximum length, in bytes, of the expected attribute data. The `AEGetAttributePtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a size variable. On return, the length, in bytes, of the data for the specified Apple event attribute. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the attribute, though `AEGetAttributePtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetAttributePtr` again.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
To get Apple event attribute data for your application to use directly, call `AEGetAttributePtr`. To get a descriptor for an Apple event attribute to pass on to another Apple Event Manager routine, call `AEGetAttributeDesc` (page 45).

Before calling `AEGetAttributePtr`, you can call the `AESizeOfAttribute` (page 98) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AEGetAttributePtr` may coerce the data, which may cause the size of the data to change.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetCoercionHandler

Gets the coercion handler for a specified descriptor type.

```
OSErr AEGetCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP *handler,
   SRefCon *handlerRefcon,
   Boolean *fromTypeIsDesc,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

toType

The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

handler

A universal procedure pointer. On return, a pointer to the specified handler, if a coercion table entry exists that exactly matches the values supplied in the parameters `fromType` and `toType`. See `AECoercionHandlerUPP` (page 168).

handlerRefcon

A pointer to a reference constant. On return, the reference constant from the coercion table entry for the specified coercion handler. The Apple Event Manager passes this reference constant to the handler each time it calls the handler. The reference constant may have a value of 0.

fromTypeIsDesc

A pointer to a Boolean value. The `AEGetCoercionHandler` function returns a value of `TRUE` in this parameter if the coercion handler expects the data as a descriptor or `FALSE`, if the coercion handler expects a pointer to the data.

isSysHandler

Specifies the coercion table to get the handler from. Pass `TRUE` to get the handler from the system coercion table or `FALSE` to get the handler from your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a coercion handler in a system coercion handler dispatch table with the goal that the handler will get called when other applications perform coercions—this won't work in Mac OS X. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

In Mac OS 7.1 through 9.x and Mac OS X version v10.2 and later, `AEGetCoercionHandler` returns `errAEHandlerNotInstalled` when there's not an exact match, even if a wildcard handler is installed that could handle the coercion. Mac OS X version v10.0.x and v10.1.x will return the wildcard handler.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetDescData

Gets the data from the specified descriptor.

```
OSErr AEGetDescData (
   const AEDesc *theAEDesc,
   void *dataPtr,
   Size maximumSize
);
```

Parameters

theAEDesc

A pointer to the descriptor to get the data from. See `AEDesc` (page 162).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes should be the same as the value you pass in the `maximumSize` parameter. On return, contains the data from the descriptor.

maximumSize

The length, in bytes, of the expected descriptor data. The `AEGetDescData` function will not return more data than you specify in this parameter. You typically determine the maximum size by calling `AEGetDescDataSize` (page 50).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

Your application can call `AEGetDescDataSize` (page 50) to get the size, in bytes, of the data in a descriptor, allocate a buffer or variable of that size, then call `AEGetDescData` to get the data.

This function works only with value descriptors created by `AECreateDesc` (page 33). You cannot get the data of an `AERecord` (page 173) or `AEDescList` (page 169), for example.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

CarbonSketch

QTCarbonShell

Declared In

`AEDataModel.h`

AEGetDescDataRange

Retrieves a specified series of bytes from the specified descriptor.

```
OSStatus AEGetDescDataRange (
   const AEDesc *dataDesc,
   void *buffer,
   Size offset,
   Size length
);
```

Parameters

dataDesc

A pointer to the descriptor to get the data from. See `AEDesc` (page 162).

buffer

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes should be at least as large as the value you pass in the `length` parameter. On return, contains the specified data from the descriptor.

offset

The zero-based offset to the data to be retrieved from the descriptor.

length

The number of bytes of contiguous data to retrieve.

Return Value

A result code. If the requested `offset` and `length` are such that they do not fit entirely within the descriptor's data, `AEGetDescDataRange` returns `errAEBufferTooSmall`. See also "Apple Event Manager Result Codes" (page 252).

Discussion

This function is valid only for value type descriptors (such as `typeUTF8Text`). You can use this function when you know the precise location of a subset of data within the descriptor. For example, if the descriptor contains a block of your private data, you might retrieve just a particular chunk you need at a known offset, representing an image, a string, or some other data type. Or if a descriptor contains an RGB color, you can access just the blue field.

When used in conjunction with `AECreateDescFromExternalPtr` (page 34), `AEGetDescDataRange` can provide greatly improved performance, especially when working with large blocks of data.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`AEDataModel.h`

AEGetDescDataSize

Gets the size, in bytes, of the data in the specified descriptor.

```
Size AEGetDescDataSize (
   const AEDesc *theAEDesc
);
```

Parameters

theAEDesc

A pointer to the descriptor to obtain the data size for. See `AEDesc` (page 162).

Return Value

Returns the size, in bytes, of the data in the specified descriptor.

Discussion
This function works only with value descriptors created by `AECreateDesc` (page 33). You cannot get the data size of an `AERecord` (page 173) or `AEDescList` (page 169), for example.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetEventHandler

Gets an event handler from an Apple event dispatch table.

```
OSErr AEGetEventHandler (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   AEEventHandlerUPP *handler,
   SRefCon *handlerRefcon,
   Boolean isSysHandler
);
```

Parameters

theAEEventClass

The event class for the desired handler. See `AEEventClass` (page 171).

theAEEventID

The event ID for the desired handler. See `AEEventID` (page 172).

handler

A universal procedure pointer. On return, a pointer to the specified handler, if a dispatch table entry exists that exactly matches the values supplied in the parameters `theAEEventClass` and `theAEEventID`.

If you use the `typeWildCard` constant for either or both of these parameters, `AEGetEventHandler` will return an error unless an entry exists that specifies `typeWildCard` in exactly the same way. For example, if you specify `typeWildCard` in both the `theAEEventClass` parameter and the `theAEEventID` parameter, the Apple Event Manager will not return the first handler for any event class and event ID in the dispatch table; instead, it will only return a handler if an entry exists that specifies type `typeWildCard` for both the event class and the event ID.

For an explanation of wildcard values, see the Discussion section for `AEInstallEventHandler` (page 65).

See `AEEventHandlerUPP` (page 171).

handlerRefcon

A pointer to a reference constant. On return, the reference constant from the dispatch table entry for the specified handler. The reference constant may have a value of 0.

isSysHandler

Specifies the Apple event dispatch table to get the handler from. Pass `TRUE` to get the handler from the system dispatch table or `FALSE` to get the handler from your application's dispatch table. See Version Notes for related information.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

In Mac OS 7.1 through 9.x and Mac OS X version v10.2 and later, `AEGetEventHandler` returns `errAEHandlerNotInstalled` when there's not an exact match, even if a wildcard handler is installed that could handle the event. Mac OS X version v10.0.x and v10.1.x will return the wildcard handler.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEGetInteractionAllowed

Gets your application's current user interaction preferences for responding to an Apple event as a server application.

```
OSErr AEGetInteractionAllowed (
   AEInteractAllowed *level
);
```

Parameters

level

A pointer to an interaction level variable. On return, the variable specifies the current user interaction level, matching one of the values described in "User Interaction Level Constants" (page 221).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The current user interaction preference for responding to an Apple event is set either by default (to `kAEInteractWithLocal`) or by a previous call to `AESetInteractionAllowed` (page 95).

For additional information on interaction level, see `AESend` (page 92) and `"AESendMode"` (page 182).

See also `AEInteractWithUser` (page 69).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEGetKeyDesc

Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event record

```
OSErr AEGetKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType desiredType,
    AEDesc *result
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter descriptor from.

theAEKeyword

A keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

desiredType

The descriptor type for the desired Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the requested Apple event parameter is not of the desired type, the Apple Event Manager attempts to coerce it to the desired type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned descriptor is the same as the descriptor type of the Apple event parameter.

See `DescType` (page 176).

result

A pointer to a descriptor. On successful return, a copy of the descriptor for the specified Apple event parameter, coerced, if necessary, to the descriptor type specified by the `desiredType` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

This function is declared as a macro that invokes `AEGetParamDesc` (page 59), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes

See `AEGetParamDesc` (page 59).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEGetKeyPtr

Gets a copy of the data for a specified Apple event parameter from an Apple event record.

```
OSErr AEGetKeyPtr (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType desiredType,
    DescType *actualType,
    void *dataPtr,
    Size maximumSize,
    Size *actualSize
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter data from.

theAEKeyword

The keyword that specifies the desired Apple event record parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 211).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetKeyPtr` attempts to coerce the data to this type. However, if the desired type is `typeWildCard`, no coercion is performed.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. Specify `NULL` if you do not care about this return value. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the parameter data. Specify `NULL` if you do not care about this return value.

maximumSize

The maximum length, in bytes, of the expected Apple event record parameter data. The `AEGetKeyPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a variable of type `Size`. On return, the length, in bytes, of the data for the specified Apple event record parameter. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the parameter, though `AEGetKeyPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetKeyPtr` again. Specify `NULL` if you do not care about this return value.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

This function is declared as a macro that invokes `AEGetParamPtr` (page 60), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See AEGetParamPtr (page 60).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEGetNthDesc

Copies a descriptor from a specified position in a descriptor list into a specified descriptor; typically used when your application needs to pass the extracted data to another function as a descriptor.

```
OSErr AEGetNthDesc (
   const AEDescList *theAEDescList,
   long index,
   DescType desiredType,
   AEKeyword *theAEKeyword,
   AEDesc *result
);
```

Parameters

theAEDescList

A pointer to the descriptor list to get the descriptor from. See AEDescList (page 169).

index

A one-based positive integer indicating the position of the descriptor to get. AEGetNthDesc returns an error if you pass zero, a negative number, or a value that is out of range.

desiredType

The desired descriptor type for the descriptor to copy. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the descriptor specified by the index parameter is not of the desired type, AEGetNthDesc attempts to coerce it to this type. However, if you pass a value of typeWildCard, no coercion is performed, and the descriptor type of the copied descriptor is the same as the descriptor type of the original descriptor.

See DescType (page 176).

theAEKeyword

A pointer to a keyword. On successful return, the keyword for the specified descriptor, if you are getting data from a list of keyword-specified descriptors; otherwise, AEGetNthDesc returns the value typeWildCard. Some keyword constants are described in "Keyword Attribute Constants" (page 209) and "Keyword Parameter Constants" (page 211). See AEKeyword (page 172).

result

A pointer to a descriptor. On successful return, a copy of the descriptor specified by the index parameter, coerced, if necessary, to the descriptor type specified by the desiredType parameter. On error, a null descriptor. If the function returns successfully, your application should call the AEDisposeDesc (page 40) function to dispose of the resulting descriptor after it has finished using it. See AEDesc (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
If the Nth descriptor in the list is itself an Apple event record and the desired type is not wildcard, record, or list, `AEGetNthDesc` will fail with an `errAECoercionFailed` error. This behavior prevents coercion problems.

You may find the `AEGetNthPtr` (page 56) function convenient for retrieving data for direct use in your application, as it includes automatic coercion.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetNthPtr

Gets a copy of the data from a descriptor at a specified position in a descriptor list; typically used when your application needs to work with the extracted data directly.

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

Parameters

theAEDescList

A pointer to the descriptor list that contains the descriptor. See `AEDescList` (page 169).

index

A one-based positive integer indicating the position in the descriptor list of the descriptor to get the data from. `AEGetNthPtr` returns an error if you pass zero, a negative number, or a value that is out of range.

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the descriptor specified by the `index` parameter is not of the desired type, `AEGetNthPtr` attempts to coerce the data to this type. If you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the copied data is the same as the descriptor type of the original descriptor.

See `DescType` (page 176).

theAEKeyword

A pointer to a keyword. On return, the keyword for the specified descriptor, if you are getting data from a list of keyword-specified descriptors; otherwise, `AEGetNthPtr` returns the value `typeWildCard`. Some keyword constants are described in "Keyword Attribute Constants" (page 209) and "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the data from the descriptor at the position in the descriptor list specified by the `index` parameter.

maximumSize

The maximum length, in bytes, of the expected data. The `AEGetNthPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a size variable. On return, the length, in bytes, of the data for the specified descriptor. If this value is larger than the value of the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the descriptor, though `AEGetNthPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetNthPtr` again.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AEGetNthPtr` function uses a buffer to return the data for a specified descriptor from a specified descriptor list. The function attempts to coerce the descriptor to the descriptor type specified by the `desiredType` parameter.

Before calling `AEGetNthPtr`, you can call the `AESizeOfNthItem` (page 100) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AESizeOfNthItem` may coerce the data, which may cause the size of the data to change. If you are using `AEGetNthPtr` to iterate through a list of descriptors of the same type with a fixed size, such as a list of descriptors of type `typeFSS`, you can get the size once, allocate a buffer, and reuse it for each call.

The order of items in an Apple event record may change after an insertion or deletion. In addition, duplicating an Apple event record is not guaranteed to preserve the item order.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell
QTMetaData

Declared In
`AEDataModel.h`

AEGetObjectAccessor

Gets an object accessor function from an object accessor dispatch table.

```
OSErr AEGetObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP *accessor,
   SRefCon *accessorRefcon,
   Boolean isSysHandler
);
```

Parameters

desiredClass

The object class of the Apple event objects located by the object accessor function to get. Pass the value `typeWildCard` to get an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the object class. Pass the value `cProperty` to get an object accessor function whose entry in an object accessor dispatch table specifies `cProperty` (a constant used to specify a property of any object class). Some other possible values are defined in "Object Class ID Constants" (page 215). See `DescType` (page 176).

containerType

The descriptor type of the token that identifies the container for the objects located by the requested accessor function. (Token is defined in `AEDisposeToken` (page 41).) Pass the value `typeWildCard` to get an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the descriptor type of the token used to specify the container type. See `DescType` (page 176).

accessor

A universal procedure pointer. On return, a pointer to the requested object accessor function, if an object accessor dispatch table entry exists that exactly matches the values supplied in the parameters `desiredClass` and `containerType`. See `OSLAccessorUPP` (page 176).

accessorRefcon

A pointer to a reference constant. On return, points to the reference constant from the object accessor dispatch table entry for the specified object accessor function. The reference constant may have a value of 0.

isSysHandler

Specifies the object accessor dispatch table to get the object accessor function from. Pass `TRUE` to get the object accessor function from the system object accessor dispatch table or `FALSE` to get the object accessor function from your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

Calling `AEGetObjectAccessor` does not remove the object accessor function from an object accessor dispatch table.

Version Notes

In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AEGetParamDesc

Gets a copy of the descriptor for a keyword-specified Apple event parameter from an Apple event or an Apple event record.

```
OSErr AEGetParamDesc (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   AEDesc *result
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter descriptor from.

theAEKeyword

A keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 211).

desiredType

The descriptor type for the desired Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the requested Apple event parameter is not of the desired type, the Apple Event Manager attempts to coerce it to the desired type. However, if you pass a value of `typeWildCard`, no coercion is performed, and the descriptor type of the returned descriptor is the same as the descriptor type of the Apple event parameter.

result

A pointer to a descriptor. On successful return, a copy of the descriptor for the specified Apple event parameter, coerced, if necessary, to the descriptor type specified by the `desiredType` parameter. On error, a null descriptor. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of the resulting descriptor after it has finished using it.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

You typically call `AEGetParamDesc` to get a descriptor for an Apple event parameter to pass on to another Apple Event Manager routine. To get Apple event parameter data for your application to use directly, call `AEGetParamPtr` (page 60).

If the actual parameter you are getting with `AEGetParamDesc` is a record, you can only request it as a `typeAERecord`, `typeAEList`, or `typeWildcard`. For any other type, `AEGetParamDesc` will return `errAECoercionFail`.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Related Sample Code

QTCarbonShell

Declared In
AEDataModel.h

AEGetParamPtr

Gets a copy of the data for a specified Apple event parameter from an Apple event or an Apple event record.

```
OSErr AEGetParamPtr (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType desiredType,
   DescType *actualType,
   void *dataPtr,
   Size maximumSize,
   Size *actualSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter data from.

theAEKeyword

The keyword that specifies the desired Apple event parameter. Some keyword constants are described in "Keyword Parameter Constants" (page 211).

desiredType

The desired descriptor type for the copied data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If the descriptor specified by the `theAEKeyword` parameter is not of the desired type, `AEGetParamPtr` attempts to coerce the data to this type. However, if the desired type is `typeWildCard`, no coercion is performed.

On return, you can determine the actual descriptor type by examining the `typeCode` parameter.

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the data pointed to by `dataPtr`. The returned type is either the same as the type specified by the `desiredType` parameter or, if the desired type was type wildcard, the true type of the descriptor. Specify `NULL` if you do not care about this return value. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

dataPtr

A pointer to a buffer, local variable, or other storage location created and disposed of by your application. The size in bytes must be at least as large as the value you pass in the `maximumSize` parameter. On return, contains the parameter data. Specify `NULL` if you do not care about this return value.

maximumSize

The maximum length, in bytes, of the expected Apple event parameter data. The `AEGetParamPtr` function will not return more data than you specify in this parameter.

actualSize

A pointer to a variable of type `Size`. On return, the length, in bytes, of the data for the specified Apple event parameter. If this value is larger than the value you passed in the `maximumSize` parameter, the buffer pointed to by `dataPtr` was not large enough to contain all of the data for the parameter, though `AEGetParamPtr` does not write beyond the end of the buffer. If the buffer was too small, you can resize it and call `AEGetParamPtr` again. Specify `NULL` if you do not care about this return value.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You should use this function only to extract data from value descriptors such as `typeUTF8Text`.

Because this function allows you to specify a desired type, it can result in coercion. When used correctly, this has the positive effect of returning the data in the desired format. However, it can have side effects you may not be expecting, such as the overhead of calls to coercion handlers. See also the Version Notes section below for possible problems with coercion.

To get Apple event parameter data for your application to use directly, call `AEGetParamPtr`. To get a descriptor for an Apple event parameter to pass on to another Apple Event Manager routine, call `AEGetParamDesc` (page 59).

Before calling `AEGetParamPtr`, you can call the `AESizeOfParam` (page 101) function to determine a size for the `dataPtr` buffer. However, unless you specify `typeWildCard` for the `desiredType` parameter, `AEGetParamPtr` may coerce the data, which may cause the size of the data to change.

In some cases, you may get improved efficiency extracting information from an Apple event with the `AEGetDescDataRange` (page 49) function.

Version Notes
Thread safe starting in Mac OS X v10.2.

If the actual parameter you are getting with `AEGetParamPtr` is a record, `AEGetParamPtr` will erroneously allow you to get the parameter as any type at all, when it really should allow only `typeAERecord`, `typeAEList`, or `typeWildcard`. For other types, it will place raw record data into the designated buffer. With AppleScript 1.1.2, it would then return `errAECoercionFail`, as expected. With AppleScript 1.3 and later, however, it returns `noErr`.

You can work around this problem by checking the returned parameter from any call to `AEGetParamPtr`. If the source type is `typeAERecord` and the type you asked for was anything other than `typeAERecord`, `typeAEList`, or `typeWildcard`, you should assume the coercion failed.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEGetRegisteredMachPort

Returns the Mach port (in the form of a `mach_port_t`) that was registered with the bootstrap server for this process.

```
mach_port_t AEGetRegisteredMachPort (
   void
);
```

Return Value
Returns a Mach message port header.

Discussion
Apple events on Mac OS X are implemented in terms of Mach messages. If your application links with the Carbon umbrella framework, it includes the HIToolbox framework, which initializes a Mach port and registers it with the run loop for the application. That port is considered public, and is used for sending and receiving Apple events.

Linking with the HIToolbox also requires that the application have a connection to the window server. To facilitate writing server processes that can send and receive Apple events, the Apple Event Manager provides the following functions (on Mac OS X only): `AEGetRegisteredMachPort`, `AEDecodeMessage` (page 37), `AESendMessage` (page 94), and `AEProcessMessage` (page 74). Daemons and other processes with no user interface can take advantage of these functions, while typical high-level applications will have no need for them.

If your code doesn't link with the HIToolbox or doesn't have a run loop, it can call `AEGetRegisteredMachPort` to register a port directly, then listen on that port for Apple events. It can use the other low-level functions to process incoming Apple events on the port and to send Apple events through it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AEGetSpecialHandler

Gets a specified handler from a special handler dispatch table.

```
OSErr AEGetSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP *handler,
   Boolean isSysHandler
);
```

Parameters

`functionClass`

The keyword for the special handler to get. You can specify any of the constants described in "Special Handler Callback Constants" (page 219). See `AEKeyword` (page 172).

`handler`

A universal procedure pointer. On return, a pointer to the specified special handler, if one exists that matches the value supplied in the `functionClass` parameter. See `AEEventHandlerUPP` (page 171).

`isSysHandler`

Specifies the special handler dispatch table to get the handler from. Pass `TRUE` to get the handler from the system special handler dispatch table or `FALSE` to get the handler from your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See also `AEInstallSpecialHandler` (page 68) and `AERemoveSpecialHandler` (page 87).

Version Notes
Thread safe starting in Mac OS X v10.2.

In Mac OS X, you should generally install all handlers in the application dispatch table. For Carbon applications running in Mac OS 8 or Mac OS 9, a special handler in the system dispatch table could reside in the system heap, where it would be available to other applications. However, this won't work in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEGetTheCurrentEvent

Gets the Apple event that is currently being handled.

```
OSErr AEGetTheCurrentEvent (
   AppleEvent *theAppleEvent
);
```

Parameters

`theAppleEvent`

A pointer to an Apple event. On return, the Apple event that is currently being handled. If no Apple event is currently being handled, `AEGetTheCurrentEvent` supplies a descriptor of descriptor type `typeNull`, which does not contain any data. See `AppleEvent` (page 175).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
In many applications, the handling of an Apple event involves one or more long chains of calls to internal functions. The `AEGetTheCurrentEvent` function makes it unnecessary for these calls to include the current Apple event as a parameter; the functions can simply call `AEGetTheCurrentEvent` to get the current Apple event when it is needed.

You can also use the `AEGetTheCurrentEvent` function to make sure that no Apple event is currently being handled. For example, suppose your application always uses an application-defined function to delete a file. That function can first call `AEGetTheCurrentEvent` and delete the file only if `AEGetTheCurrentEvent` returns a null descriptor (that is, only if no Apple event is currently being handled).

Special Considerations

This function is not thread-safe and should only be called on the main thread.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEInitializeDesc

Initializes a new descriptor.

```
void AEInitializeDesc (
   AEDesc *desc
);
```

Parameters

desc

A pointer to a new descriptor. See AEDesc (page 162).

Discussion

The function sets the type of the descriptor to typeNull and sets the data handle to NULL. If you need to initialize a descriptor that already has some data in it, use AEDisposeDesc (page 40) to deallocate the memory and initialize the descriptor.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEDataModel.h

AEInstallCoercionHandler

Installs a coercion handler in either the application or system coercion handler dispatch table.

```
OSErr AEInstallCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP handler,
   SRefCon handlerRefcon,
   Boolean fromTypeIsDesc,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See DescType (page 176).

toType

The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197).

If there was already an entry in the specified coercion handler table for the same source descriptor type and result descriptor type, the existing entry is replaced. See DescType (page 176).

handler

A universal procedure pointer to the coercion handler function to install. See AECoercionHandlerUPP (page 168).

handlerRefcon

A reference constant. The Apple Event Manager passes this value to the handler each time it calls it. If your handler doesn't require a reference constant, pass 0 for this parameter.

fromTypeIsDesc

Specifies the form of the data to coerce. Pass `TRUE` if the coercion handler expects the data as a descriptor or `FALSE` if the coercion handler expects a pointer to the data. The Apple Event Manager can provide a pointer to data more efficiently than it can provide a descriptor, so all coercion functions should accept a pointer to data if possible.

isSysHandler

Specifies the coercion table to add the handler to. Pass `TRUE` to add the handler to the system coercion table or `FALSE` to add the handler to your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Before using `AEInstallCoercionHandler` to install a handler for a particular descriptor type, you can use the `AEGetCoercionHandler` (page 47) function to determine whether the table already contains a coercion handler for that type.

Version Notes
See the Version Notes section for the `AECoercePtr` (page 30) function for information on when to use descriptor-based versus pointer-based coercion handlers starting in Mac OS X version 10.2.

Thread safe starting in Mac OS X v10.2.

Your application should not install a coercion handler in a system coercion handler dispatch table with the goal that the handler will get called when other applications perform coercions—this won't work in Mac OS X. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEInstallEventHandler

Adds an entry for an event handler to an Apple event dispatch table.

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

Parameters

theAEEventClass

The event class for the Apple event or events to dispatch to this event handler. The Discussion section describes interactions between this parameter and the `theAEEventID` parameter. See `AEEventClass` (page 171).

theAEEventID

The event ID for the Apple event or events to dispatch to this event handler. The Discussion section describes interactions between this parameter and the `theAEEventClass` parameter. See `AEEventID` (page 172).

handler

A universal procedure pointer to the Apple event handler function to install. See `AEEventHandlerUPP` (page 171).

handlerRefcon

A reference constant. The Apple Event Manager passes this value to the handler each time it calls it. If your handler doesn't require a reference constant, pass 0 for this parameter.

isSysHandler

Specifies the Apple event dispatch table to add the handler to. Pass `TRUE` to add the handler to the system dispatch table or `FALSE` to add the handler to your application's dispatch table. See Version Notes for related information.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

The parameters `theAEEventClass` and `theAEEventID` specify the event class and event ID of the Apple events handled by the handler for this dispatch table entry. If there is already an entry in the specified dispatch table for the same event class and event ID, it is replaced. For these parameters, you must provide one of the following combinations:

- the event class and event ID of a single Apple event to dispatch to the handler (for example, an event class of `kAECoreSuite` and an event ID of `kAEDelete` so that a specific kind of `delete` event is dispatched to the handler)
- the `typeWildCard` constant for `theAEEventClass` and an event ID for `theAEEventID`, which indicates that Apple events from all event classes whose event IDs match `theAEEventID` should be dispatched to the handler (for example, an event class of `typeWildCard` and an event ID of `kAEDelete` so that for all event classes, the `delete` event is dispatched to the handler)
- an event class for `theAEEventClass` and the `typeWildCard` constant for `theAEEventID`, which indicates that all events from the specified event class should be dispatched to the handler (for example, an event class of `kAECoreSuite` and an event ID of `typeWildCard` so that all events for the core suite are dispatched to the handler)
- the `typeWildCard` constant for both the `theAEEventClass` and `theAEEventID` parameters, which indicates that all Apple events should be dispatched to the handler

If you use the `typeWildCard` constant for either the `theAEEventClass` or the `theAEEventID` parameter (or for both parameters), the corresponding handler must return the error `errAEEventNotHandled` if it does not handle a particular event.

If an Apple event dispatch table contains one entry for an event class and a specific event ID, and also contains another entry that is identical except that it specifies a wildcard value for either the event class or the event ID, the Apple Event Manager dispatches the more specific entry. For example, if an Apple event dispatch table includes one entry that specifies the event class as `kAECoreSuite` and the event ID as `kAEDelete`, and another entry that specifies the event class as `kAECoreSuite` and the event ID as `typeWildCard`, the Apple Event Manager dispatches the Apple event handler associated with the entry that specifies the event ID as `kAEDelete`.

In addition to the Apple event handler dispatch tables, applications can add entries to special handler dispatch tables, as described in "Managing Special Handler Dispatch Tables" (page 20).

Version Notes
Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
CarbonSketch
QTCarbonShell

Declared In
`AppleEvents.h`

AEInstallObjectAccessor

Adds or replaces an entry for an object accessor function to an object accessor dispatch table.

```
OSErr AEInstallObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP theAccessor,
   SRefCon accessorRefcon,
   Boolean isSysHandler
);
```

Parameters

desiredClass
The object type of the Apple event objects located by this accessor. See `DescType` (page 176).

containerType
The type of the token whose objects are accessed by this accessor. (Token is defined in `AEDisposeToken` (page 41).) The accessor function finds objects in containers specified by tokens of this type. See `DescType` (page 176).

theAccessor
A universal procedure pointer to the object accessor function to install. See `OSLAccessorUPP` (page 176).

accessorRefcon
A reference constant the Apple Event Manager passes to the object accessor function whenever it calls the function. If your object accessor function doesn't require a reference constant, pass 0 for this parameter. To change the value of the reference constant, you must call `AEInstallObjectAccessor` again.

isSysHandler
Specifies the object accessor dispatch table to add the entry to. Pass `TRUE` to add the entry to the system object accessor dispatch table or `FALSE` to add the entry to your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AEInstallObjectAccessor` function adds or replaces an entry to either the application or system object accessor dispatch table.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

If your Carbon application running in Mac OS 8 or OS 9 installs a system object accessor function in its application heap, rather than in the system heap, you must call `AERemoveObjectAccessor` (page 86) to remove the function before your application terminates.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEInstallSpecialHandler

Installs a callback function in a special handler dispatch table.

```
OSErr AEInstallSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

functionClass

A value that specifies the type of handler to install. You can use any of the constants defined in "Special Handler Callback Constants" (page 219).

If there is already an entry in the specified special handler dispatch table for the value you specify in this parameter, it is replaced.

See `AEKeyword` (page 172).

handler

A universal procedure pointer to the special handler to install. See `AEEventHandlerUPP` (page 171).

isSysHandler

Specifies the special handler dispatch table to add the handler to. Pass `TRUE` to add the handler to the system special handler dispatch table or `FALSE` to add the handler to your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
An Apple event special handler dispatch table contains entries with a function class keyword, the address of the handler function that handles the Apple events indicated by the keyword, and a reference constant. Depending on which handlers you choose to install, a special handler dispatch table can have entries for any of the following:

- a predispatch handler (an Apple event handler that the Apple Event Manager calls immediately before it dispatches an Apple event)
- up to one each of the callback functions described in "Object Callback Functions" (page 140) these functions, such as an object comparison function and an object-counting function, can be installed with `AEInstallSpecialHandler` or with the `AEInstallObjectAccessor` (page 67) function

See also `AEGetSpecialHandler` (page 62) and `AERemoveSpecialHandler` (page 87).

Version Notes
Thread safe starting in Mac OS X v10.2.

For Carbon applications running in Mac OS 8 or Mac OS 9, a handler in the system special handler dispatch table should reside in the system heap, where it may be available to other applications. If you put your system handler code in your application heap, be sure to use `AERemoveSpecialHandler` to remove the handler when your application quits. Otherwise, your handler will still have an entry in the system dispatch table with a pointer a handler that no longer exists. Another application may dispatch an Apple event that attempts to call your handler, leading to a system crash.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEInteractWithUser

Initiates interaction with the user when your application is a server application responding to an Apple event.

```
OSErr AEInteractWithUser (
   SInt32 timeOutInTicks,
   NMRecPtr nmReqPtr,
   AEIdleUPP idleProc
);
```

Parameters

timeOutInTicks
The amount of time (in ticks) that your handler is willing to wait for a response from the user. You can specify a number of ticks or use one of the constants defined in "Timeout Constants" (page 221).

nmReqPtr
A pointer to a Notification Manager record provided by your application. You can specify `NULL` for this parameter to get the default notification handling provided by the Apple Event Manager. See the Notification Manager documentation for a description of the `NMRecPtr` data type.

idleProc
A universal procedure pointer to your application's idle function, which handles events while waiting for the Apple Event Manager to return control. See `AEIdleUPP` (page 172).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). The `AEInteractWithUser` function returns the `errAENoUserInteraction` result code if the user interaction preferences don't allow user interaction. If `AEInteractWithUser` returns the `noErr` result code, then your application is in the foreground and is free to interact with the user.

Discussion
Your application should call the `AEInteractWithUser` function before displaying a dialog box or alert box or otherwise interacting with the user in response to an Apple event. The `AEInteractWithUser` function checks whether the client application set the `kAENeverInteract` flag for the current Apple event, if any, and if so, returns an error. If not, then `AEInteractWithUser` checks the server application's preference set by the `AESetInteractionAllowed` (page 95) function and compares it against the source of the Apple event—that is, whether it came from the same application, another process on the same computer, or a process running on another computer.

If the user interaction preference settings permit the application to come to the foreground, this function brings your application to the front, either directly or by posting a notification request.

Your application should normally pass a notification record in the `nmReqPtr` parameter rather than specifying `NULL` for default notification handling. If you specify `NULL`, the Apple Event Manager looks for an application icon with the ID specified by the application's bundle (`'BNDL'`) resource and the application's file reference (`'FREF'`) resource. The Apple Event Manager first looks for an `'SICN'` resource with the specified ID if it can't find an `'SICN'` resource, it looks for the `'ICN#'` resource and compresses the icon to fit in the menu bar. The Apple Event Manager won't look for any members of an icon family other than the icon specified in the `'ICN#'` resource.

If the application doesn't have `'SICN'` or `'ICN#'` resources, or if it doesn't have a file reference resource, the Apple Event Manager passes no icon to the Notification Manager, and no icon appears in the upper-right corner of the screen. Therefore, if you want to display any icon other than those of type `'SICN'` or `'ICN#'`, you must specify a notification record as the second parameter to the AEInteractWithUser function.

If you want the Notification Manager to use a color icon when it posts a notification request, you should provide a Notification Manager record that specifies a `'cicn'` resource.

For additional information on interaction level, see `AESend` (page 92) and "`AESendMode`" (page 182).

See also `AESetInteractionAllowed` (page 95) and `AEGetInteractionAllowed` (page 52).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEManagerInfo

Provides information about the version of the Apple Event Manager currently available or the number of processes that are currently recording Apple events.

```
OSErr AEManagerInfo (
   AEKeyword keyWord,
   long *result
);
```

Parameters

keyWord

A value that determines the kind of information the function supplies in the `result` parameter.

Pass the value `keyAERecorderCount` to obtain the number of processes that are currently recording Apple events.

Pass the value `keyAEVersion` to obtain version information for the Apple Event Manager, in `NumVersion` format.

Some keyword constants are defined in "Keyword Parameter Constants" (page 211).

See `AEKeyword` (page 172).

result

A pointer to a long value. On return, provides information that depends on what you pass in the `keyword` parameter.

If you pass `keyAERecorderCount`, `result` specifies the number of processes that are currently recording Apple events.

If you pass `keyAEVersion`, `result` supplies version information for the Apple Event Manager, in a format that matches the `'vers'` resource.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
For recordable applications, the information provided by `AEManagerInfo` may be useful when the application is responding to Apple events that it sends to itself.

For information on determining whether the Apple Event Manager is available, see the Apple Event Manager Gestalt Selector, described in *Inside Mac OS X: Gestalt Manager Reference*.

Version Notes
Thread safe starting in Mac OS X v10.2.

The `AEManagerInfo` function is available only in version 1.01 and later of the Apple Event Manager.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEObjectInit

Initializes the Object Support Library.

```
OSErr AEObjectInit (
   void
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You must call this function before calling any of the Apple Event Manager functions that describe or manipulate Apple event objects.

You should call the `AEObjectInit` function to initialize the Apple Event Manager functions that handle object specifiers and Apple event objects.

Version Notes
To make these functions available to your application with version 1.01 and earlier versions of the Apple Event Manager, you must also link the Apple Event Object Support Library with your application when you build it. For more information, see the Version Notes section for the AppleScript Gestalt Selector described in *Inside Mac OS X: Gestalt Manager Reference* and the function `AERemoveSpecialHandler` (page 87).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEPrintDescToHandle

Provides a pretty printer facility for displaying the contents of Apple event descriptors.

```
OSStatus AEPrintDescToHandle (
   const AEDesc *desc,
   Handle *result
);
```

Parameters

desc

A pointer to a descriptor containing the information to be printed. See `AEDesc` (page 162).

result

A pointer to a location for a new `Handle` data type. On return, contains a new handle allocated by the Memory Manager.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The data handle returned in the *result* parameter contains a text string formatted using the "AEBuild" syntax. This string is useful for looking at the contents of Apple events sent by other applications and for debugging your own descriptors.

`AEPrintDescToHandle` prints the contents of `AEDesc`, `AERecord`, and `AEDescList` descriptors in a format that is suitable for input to `AEBuildDesc` (page 26). `AEPrintDescToHandle` also attempts display coerced Apple event records as the coerced record type instead of as the original type. Any data structures that cannot be identified are displayed as hexadecimal data.

AEPrintDescToHandle prints the contents of Apple events in a slightly different format. For these events, the event class and event ID appear at the beginning of the output string, followed by the contents of the event enclosed in curly braces. In addition, each attribute is printed with its four-character identifier and preceded by an ampersand character. You cannot use the output string to recreate the Apple event from AEBuildAppleEvent (page 24).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AEProcessAppleEvent

Calls the handler, if one exists, for a specified Apple event.

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

Parameters

theEventRecord

A pointer to the event record for the Apple event to process. See the Event Manager documentation for a description of the EventRecord data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). This is the error result from the Apple event handler (or errAEHandlerNotFound). In most cases your application should ignore this error because it will be seen by the Apple event sender as the keyErrorNumber parameter in the reply.

Discussion
After receiving a high-level event (and optionally determining whether it is a type of high-level event other than an Apple event that your application might support), your application typically calls the AEProcessAppleEvent function to determine the type of Apple event received and call the corresponding handler. Your application should always handle high-level events immediately, or the Apple Event Manager may return the event to the sending application with the errAEEventNotHandled result code.

The AEProcessAppleEvent function looks first in the application's special handler dispatch table for an entry that was installed by the AEInstallSpecialHandler (page 68) function with the constant keyPreDispatch. If the application's special handler dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, AEProcessAppleEvent looks in the application's Apple event dispatch table for an entry that matches the event class and event ID of the specified Apple event. You install handlers in the application's dispatch table with the AEInstallEventHandler (page 65) function.

If the application's Apple event dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, the AEProcessAppleEvent function looks in the system special handler dispatch table for an entry that was installed with the constant keyPreDispatch. If the system special handler dispatch table does not include such a handler or if the handler returns errAEEventNotHandled, AEProcessAppleEvent looks in the system Apple event dispatch table for an entry that matches the event class and event ID of the specified Apple event.

If the system Apple event dispatch table does not include such a handler, the Apple Event Manager returns the result code errAEEventNotHandled to the server (or target) application and, if the client application is waiting for a reply, to the client application.

If AEProcessAppleEvent finds an entry in one of the dispatch tables that matches the event class and event ID of the specified Apple event, it calls the corresponding handler.

If an Apple event dispatch table contains one entry for an event class and a specific event ID, and also contains another entry that specifies a wildcard value for either the event class or the event ID, the Apple Event Manager uses the more specific entry. For example, if one entry specifies an event class of kAECoreSuite and an event ID of kAEDelete and another entry specifies an event class of kAECoreSuite and an event ID of typeWildCard, the Apple Event Manager will dispatch an Apple event with an event ID of kAEDelete to the handler from the entry that specifies the event ID as kAEDelete.

Version Notes
Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
Simple DrawSprocket

Declared In
AEInteraction.h

AEProcessMessage

Decodes and dispatches a low level Mach message event to an event handler, including packaging and returning the reply to the sender.

```
OSStatus AEProcessMessage (
   mach_msg_header_t *header
);
```

Parameters

header

A pointer to the received Mach message that should be processed. The contents of the message header are invalid after calling this method.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The Apple Event Manager provides the following functions (on Mac OS X only) for working with Apple events at a lower level: AEGetRegisteredMachPort (page 61), AEDecodeMessage (page 37), AESendMessage (page 94), and AEProcessMessage. See the descriptions for those functions for more information on when you might use them.

If your daemon or other code has initialized a Mach port and is listening on it for Apple events and other messages, it can call AEProcessMessage to handle any incoming events it identifies as Apple events, while handling other types of events itself. AEProcessMessage will dispatch the event to an event handler (by calling AEDecodeMessage for you) and package and return the reply to the sender, simplifying handling for your code.

The Apple Event Manager reserves Mach message IDs in the range 0 to 999 for its own use. AEProcessMessage returns a paramErr result code if the Mach message did not contain an Apple event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEMach.h

AEPutArray

Inserts the data for an Apple event array into a descriptor list, replacing any previous descriptors in the list.

```
OSErr AEPutArray (
   AEDescList *theAEDescList,
   AEArrayType arrayType,
   const AEArrayData *arrayPtr,
   DescType itemType,
   Size itemSize,
   long itemCount
);
```

Parameters

theAEDescList

A pointer to the descriptor list to put the Apple event array into. If there are any descriptors already in the descriptor list, they are replaced. If the array type is kAEKeyDescArray, theAEDescList must point to an Apple event record; otherwise, it can point to either a descriptor list or an Apple event record.

If you pass a pointer to a factored descriptor list, created by calling the AECreateList (page 35) function, each array item in the array pointed to by the arrayPtr parameter must include the data that is common to all the descriptors in the list. The Apple Event Manager automatically isolates the common data you specified in the call to AECreateList. A factored descriptor list is described in the Discussion section.

See AEDescList (page 169).

arrayType

The Apple event array type to create. Pass a value specified by one of the constants described in "Data Array Constants" (page 196). See AEArrayType (page 168).

arrayPtr

A pointer to a buffer, local variable, or other storage location, created and disposed of by your application, that contains the array to put into the descriptor list. See AEArrayData (page 161).

itemType

For arrays of type kAEDataArray, kAEPackedArray, or kAEHandleArray, the descriptor type of the array items to create. Use one of the constants described in "Descriptor Type Constants" (page 197), such as typeLongInteger. You don't need to specify an item type for arrays of type kAEDescArray or kAEKeyDescArray because the data is already stored in descriptors which contain a descriptor type. See DescType (page 176).

itemSize

For arrays of type kAEDataArray or kAEPackedArray, the size (in bytes) of the array items to create. You don't need to specify an item size for arrays of type kAEDescArray, kAEKeyDescArray, or kAEHandleArray because their descriptors (though not the data they point to) have a known size.

itemCount

The number of elements in the array.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
A factored descriptor list is one in which the Apple Event Manager automatically isolates the data that is common to all the elements of the list so that the common data only appears in the list once. To create a factored descriptor list, you call the `AECreateList` (page 35) function and specify the data that is common to all elements in the descriptor array.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutAttributeDesc

Adds a descriptor and a keyword to an Apple event as an attribute.

```
OSErr AEPutAttributeDesc (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   const AEDesc *theAEDesc
);
```

Parameters

theAppleEvent

A pointer to the Apple event to add an attribute to. See the `AppleEvent` (page 175) data type.

theAEKeyword

The keyword for the attribute to add. If the Apple event already includes an attribute with this keyword, the attribute is replaced.

Some keyword constants are described in "Keyword Attribute Constants" (page 209).

See `AEKeyword` (page 172).

theAEDesc

A pointer to the descriptor to assign to the attribute. The descriptor type of the specified descriptor should match the defined descriptor type for that attribute. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AEPutAttributeDesc` function takes a descriptor and a keyword and adds them to an Apple event as an attribute. If the descriptor type required for the attribute is different from the descriptor type of the descriptor, the Apple Event Manager attempts to coerce the descriptor into the required type, with one exception: the Apple Event Manager does not attempt to coerce the data for an address attribute, thereby allowing applications to use their own address types.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutAttributePtr

Adds a pointer to data, a descriptor type, and a keyword to an Apple event as an attribute.

```
OSErr AEPutAttributePtr (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to add an attribute to. See the AppleEvent (page 175) data type.

theAEKeyword

The keyword for the attribute to add. If the Apple event already includes an attribute with this keyword, the attribute is replaced.

Some keyword constants are described in "Keyword Attribute Constants" (page 209).

See AEKeyword (page 172).

typeCode

The descriptor type for the attribute to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See DescType (page 176).

dataPtr

A pointer to the data for the attribute to add.

dataSize

The length, in bytes, of the data for the attribute to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutDesc

Adds a descriptor to any descriptor list, possibly replacing an existing descriptor in the list.

```
OSErr AEPutDesc (
   AEDescList *theAEDescList,
   long index,
   const AEDesc *theAEDesc
);
```

Parameters

theAEDescList

A pointer to the descriptor list to add a descriptor to. See `AEDescList` (page 169).

index

A one-based positive integer indicating the position to insert the descriptor at. If there is already a descriptor in the specified position, it is replaced.

You can pass a value of zero or count + 1 to add the descriptor at the end of the list. `AEPutDesc` returns an error (`AEIllegalIndex`) if you pass a negative number or a value that is out of range.

theAEDesc

A pointer to the descriptor to add to the list. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutKeyDesc

Inserts a descriptor and a keyword into an Apple event record as an Apple event parameter.

```
OSErr AEPutKeyDesc (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    const AEDesc *theAEDesc
);
```

Parameters

theAERecord

A pointer to the Apple event record to add a parameter to.

theAEKeyword

The keyword specifying the parameter to add. If the Apple event record already has a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 211).

See `AEKeyword` (page 172).

theAEDesc

A pointer to the descriptor for the parameter to add. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function is declared as a macro that invokes `AEPutParamDesc` (page 80), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AEPutParamDesc` (page 80).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEPutKeyPtr

Inserts data, a descriptor type, and a keyword into an Apple event record as an Apple event parameter.

```
OSErr AEPutKeyPtr (
    AERecord *theAERecord,
    AEKeyword theAEKeyword,
    DescType typeCode,
    const void *dataPtr,
    Size dataSize
);
```

Parameters

theAERecord
A pointer to the Apple event record to add a parameter to.

theAEKeyword
The keyword for the parameter to add. If the Apple event record already includes a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 211).

See `AEKeyword` (page 172).

typeCode
The descriptor type for the parameter to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr
A pointer to the data for the parameter to add.

dataSize
The length, in bytes, of the data for the parameter to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function is declared as a macro that invokes `AEPutParamPtr` (page 80), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AEPutParamPtr` (page 80).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutParamDesc

Inserts a descriptor and a keyword into an Apple event or Apple event record as an Apple event parameter.

```
OSErr AEPutParamDesc (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   const AEDesc *theAEDesc
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add a parameter to. See the AppleEvent (page 175) data type.

theAEKeyword
The keyword specifying the parameter to add. If the Apple event already has a parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 211).

See AEKeyword (page 172).

theAEDesc
A pointer to the descriptor for the parameter to add. See AEDesc (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEPutParamPtr

Inserts data, a descriptor type, and a keyword into an Apple event or Apple event record as an Apple event parameter.

```
OSErr AEPutParamPtr (
   AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAppleEvent
A pointer to the Apple event to add a parameter to. See the AppleEvent (page 175) data type.

theAEKeyword

The keyword for the parameter to add. If the Apple event already includes an parameter with this keyword, the parameter is replaced.

Some keyword constants are described in "Keyword Parameter Constants" (page 211).

See `AEKeyword` (page 172).

typeCode

The descriptor type for the parameter to add. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to the data for the parameter to add.

dataSize

The length, in bytes, of the data for the parameter to add.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEPutPtr

Inserts data specified in a buffer into a descriptor list as a descriptor, possibly replacing an existing descriptor in the list.

```
OSErr AEPutPtr (
   AEDescList *theAEDescList,
   long index,
   DescType typeCode,
   const void *dataPtr,
   Size dataSize
);
```

Parameters

theAEDescList

A pointer to the descriptor list to add a descriptor to. See `AEDescList` (page 169).

index

A one-based positive integer indicating the position to insert the descriptor at. If there is already a descriptor in the specified position, it is replaced.

You can pass a value of zero or count + 1 to add the descriptor at the end of the list. `AEPutPtr` returns an error (`AEIllegalIndex`) if you pass a negative number or a value that is out of range.

typeCode

The descriptor type for the descriptor to be put into the list. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to the data for the descriptor to add.

dataSize

The length, in bytes, of the data for the descriptor to add.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
QTCarbonShell

Declared In
`AEDataModel.h`

AERemoteProcessResolverGetProcesses

Returns an array of objects containing information about processes running on a remote machine.

```
CFArrayRef AERemoteProcessResolverGetProcesses (
   AERemoteProcessResolverRef ref,
   CFStreamError *outError
);
```

Parameters

ref

The `AERemoteProcessResolverRef` (page 173) to query. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 36).

outError

If the function result is `NULL`, `outError` contains information about the failure. See the Core Foundation Reference Documentation for a description of the `CFStreamError` data type.

Return Value
In the case of an error, returns `NULL`, in which case the `outError` parameter provides error information. If successful, returns a `CFArrayRef` of `CFDictionaryRef` objects containing information about the discovered remote processes. Each dictionary contains the URL of a remote application and its human readable name; it may also contain a `CFNumberRef` specifying a user ID for the application, if it has one; and it may also contain a `CFNumberRef` specifying the process ID for the process. The array is owned by the resolver, so you must retain it before disposing of the resolver object itself. For information on the keys for getting information from the dictionary, see "Remote Process Dictionary Keys" (page 218).

Discussion
You first call `AECreateRemoteProcessResolver` (page 36) to obtain a reference to a resolver object you can use to obtain a list of processes running on a specified remote machine. See the description for that function for additional information. You then pass that reference to `AERemoteProcessResolverGetProcesses` to get an array of objects containing information about the discovered remote processes.

If the resolver was not previously scheduled for execution (by a call to the `AERemoteProcessResolverScheduleWithRunLoop` (page 83) function), `AERemoteProcessResolverGetProcesses` will block until the resulting array is available or an error occurs. If the resolver was previously scheduled but had not yet completed fetching the array, this call will block until the resolver does complete.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AERemoteProcessResolverScheduleWithRunLoop

Schedules a resolver for execution on a given run loop in a given mode.

```
void AERemoteProcessResolverScheduleWithRunLoop (
   AERemoteProcessResolverRef ref,
   CFRunLoopRef runLoop,
   CFStringRef runLoopMode,
   AERemoteProcessResolverCallback callback,
   const AERemoteProcessResolverContext *ctx
);
```

Parameters

ref

The `AERemoteProcessResolverRef` (page 173) to query. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 36).

runLoop

The run loop on which to schedule resolution of remote processes. For information on run loops, see Introduction to Run Loops. See the Core Foundation Reference Documentation for a description of the `CFRunLoop` data type.

runLoopMode

Specifies the run loop mode. See Input Modes for information on available modes. See the Core Foundation Reference Documentation for a description of the `CFStringRef` data type.

callback

A callback function to be executed when the resolver completes. See `AERemoteProcessResolverCallback` (page 148) for information on the callback definition.

ctx

Optionally supplies information of use while resolving remote processes. If this parameter is not `NULL`, the info field of this structure is passed to the callback function (otherwise, the info parameter to the `callback` function will explicitly be `NULL`). See `AERemoteProcessResolverContext` (page 163) for a description of this data type.

Discussion
Schedules a resolver for execution on a given run loop in a given mode. The resolver will move through various internal states as long as the specified run loop is run. When the resolver completes, either with success or with an error condition, the callback is executed. There is no explicit unschedule of the resolver; you must dispose of it to remove it from the run loop.

Version Notes
Thread safe starting in Mac OS X v10.3.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AERemoveCoercionHandler

Removes a coercion handler from a coercion handler dispatch table.

```
OSErr AERemoveCoercionHandler (
   DescType fromType,
   DescType toType,
   AECoercionHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

fromType

The descriptor type of the data coerced by the handler. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

toType

The descriptor type of the resulting data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

handler

A universal procedure pointer to the coercion handler to remove. Although the parameters `fromType` and `toType` are sufficient to identify the handler, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the event class and event ID to identify the handler. See `AECoercionHandlerUPP` (page 168).

isSysHandler

Specifies the coercion table to remove the handler from. Pass `TRUE` to remove the handler from the system coercion table or `FALSE` to remove the handler from your application's coercion table. Use of the system coercion table is not recommended.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Use of system coercion tables is not recommended. For more information, see "Writing and Installing Coercion Handlers" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoveEventHandler

Removes an event handler entry from an Apple event dispatch table.

```
OSErr AERemoveEventHandler (
   AEEventClass theAEEventClass,
   AEEventID theAEEventID,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

theAEEventClass

The event class for the handler to remove. See `AEEventClass` (page 171).

theAEEventID

The event ID for the handler to remove. See `AEEventID` (page 172).

handler

A universal procedure pointer to the handler to remove. Although the parameters `theAEEventClass` and `theAEEventID` are sufficient to identify the handler, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the event class and event ID to identify the handler.

If you use the `typeWildCard` constant for either or both of the event class and event ID parameters, `AERemoveEventHandler` will return an error unless an entry exists that specifies `typeWildCard` in exactly the same way. For example, if you specify `typeWildCard` in both the `theAEEventClass` parameter and the `theAEEventID` parameter, `AERemoveEventHandler` will not remove the first handler for any event class and event ID in the dispatch table; instead, it will only remove a handler if an entry exists that specifies type `typeWildCard` for both the event class and the event ID.

For an explanation of wildcard values, see the Discussion section for `AEInstallEventHandler` (page 65).

See `AEEventHandlerUPP` (page 171).

isSysHandler

Specifies the Apple event dispatch table to remove the handler from. Pass `TRUE` to remove the handler from the system dispatch table or `FALSE` to remove the handler from your application's dispatch table. See Version Notes for related information.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes

Thread safe starting in Mac OS X v10.2.

Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AppleEvents.h`

AERemoveObjectAccessor

Removes an object accessor function from an object accessor dispatch table.

```
OSErr AERemoveObjectAccessor (
   DescType desiredClass,
   DescType containerType,
   OSLAccessorUPP theAccessor,
   Boolean isSysHandler
);
```

Parameters

desiredClass

The object class of the Apple event objects located by the object accessor function to remove. Pass the value `typeWildCard` to remove an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the object class. Pass the value `cProperty` to remove an object accessor function whose entry in an object accessor dispatch table specifies `cProperty` (a constant used to specify a property of any object class). Some other possible values are defined in "Object Class ID Constants" (page 215). See `DescType` (page 176).

containerType

The descriptor type of the token that identifies the container for the objects located by the object accessor function to remove. (Token is defined in `AEDisposeToken` (page 41).) Pass the value `typeWildCard` to remove an object accessor function whose entry in an object accessor dispatch table specifies `typeWildCard` as the descriptor type of the token used to specify the container type. See `DescType` (page 176).

theAccessor

A universal procedure pointer to the special handler to remove. Although the `functionClass` parameter is sufficient to identify the handler to remove, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the function class to identify the handler. A universal procedure pointer (UPP) to the object accessor function to remove. Although the parameters `desiredClass` and `containerType` are sufficient to identify the function to remove, you can identify the function explicitly by providing a UPP in this parameter. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the desired class and container type. See `OSLAccessorUPP` (page 176).

isSysHandler

Specifies the object accessor dispatch table to remove the object accessor function from. Pass `TRUE` to remove the object accessor function from the system object accessor dispatch table or `FALSE` to remove the object accessor function from your application's object accessor dispatch table. Use of the system object accessor dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes

In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AERemoveSpecialHandler

Removes a handler from a special handler dispatch table.

```
OSErr AERemoveSpecialHandler (
   AEKeyword functionClass,
   AEEventHandlerUPP handler,
   Boolean isSysHandler
);
```

Parameters

functionClass

The keyword for the special handler to remove. Pass one of the constants described in "Special Handler Callback Constants" (page 219). See `AEKeyword` (page 172).

handler

A universal procedure pointer to the special handler to remove. Although the `functionClass` parameter is sufficient to identify the handler to remove, you can identify the handler explicitly as a safeguard. If you pass `NULL` for this parameter, the Apple Event Manager relies solely on the function class to identify the handler. See `AEEventHandlerUPP` (page 171).

isSysHandler

Specifies the special handler dispatch table to remove the handler from. Pass `TRUE` to remove the handler from the system special handler dispatch table or `FALSE` to remove the handler from your application's special handler dispatch table. Use of the system special handler dispatch table is not recommended.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

See also `AEInstallSpecialHandler` (page 68) and `AEGetSpecialHandler` (page 62).

Version Notes

Thread safe starting in Mac OS X v10.2.

Your application should not install a special handler in a system dispatch table with the goal that the handler will get called when other applications receive events—this won't work in Mac OS X.

In some previous versions of the Mac OS, applications might have reason to disable, within the application only, all Apple Event Manager functions that support Apple event objects—that is, all the functions available to an application as a result of linking the Object Support Library (OSL) and calling the `AEObjectInit` (page 71) function.

To disable the OSL, you should pass the keyword `keySelectProc` in the `functionClass` parameter, `NULL` in the `handler` parameter, and `FALSE` in the `isSysHandler` parameter. An application that expects its copy of the OSL to move after it is installed—for example, an application that keeps it in a stand-alone code resource—would need to disable the OSL. When an application calls `AEObjectInit` to initialize the OSL, the OSL installs the addresses of its functions as extensions to the pack. If those functions move, the addresses become invalid.

Once you have called the `AERemoveSpecialHandler` function to disable the OSL, subsequent calls by your application to any of the Apple Event Manager functions that support Apple event objects will return errors. To initialize the OSL after disabling it with the `AERemoveSpecialHandler` function, your application must call `AEObjectInit` again.

If you expect to initialize the OSL and disable it several times, you should call `AERemoveObjectAccessor` to remove your application's object accessor functions from your application's object accessor dispatch table before you call `AERemoveSpecialHandler`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEReplaceDescData

Copies the specified data into the specified descriptor, replacing any previous data.

```
OSErr AEReplaceDescData (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   AEDesc *theAEDesc
);
```

Parameters

typeCode
Specifies the descriptor type of the data pointed to by `dataPtr`. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr
A pointer to the data to store in the specified descriptor.

dataSize
The size, in bytes, of the data pointed to by the `dataSize` parameter.

theAEDesc
A pointer to a descriptor. On return, contains the copied data. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEResetTimer

Resets the timeout value for an Apple event to its starting value.

```
OSErr AEResetTimer (
   const AppleEvent *reply
);
```

Parameters

reply

A pointer to the default reply for an Apple event, provided by the Apple Event Manager. See `AppleEvent` (page 175).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

The `AEResetTimer` function resets the timeout value for an Apple event to its starting value. A server application can call this function when it knows it cannot fulfill a client application's request (either by returning a result or by sending back a reply Apple event) before the client application is due to time out.

When your application calls `AEResetTimer`, the Apple Event Manager for the server application uses the default reply to send a Reset Timer event to the client application the Apple Event Manager for the client application's computer intercepts this Apple event and resets the client application's timer for the Apple event. (The Reset Timer event is never dispatched to a handler, so the client application does not need a handler for it.)

Version Notes

Prior to Mac OS X version 10.3, calling `AEResetTimer` did not reset the timeout value.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEInteraction.h`

AEResolve

Resolves an object specifier.

```
OSErr AEResolve (
   const AEDesc *objectSpecifier,
   short callbackFlags,
   AEDesc *theToken
);
```

Parameters

objectSpecifier

A pointer to the object specifier to resolve. See `AEDesc` (page 162).

callbackFlags

A value that determines what additional assistance, if any, your application can give the Apple Event Manager when it parses the object specifier. The value is specified by adding the desired constants described in "Callback Constants for the AEResolve Function" (page 187). Most applications use `kAEIDoMinimum`.

theToken

A pointer to a descriptor. On return, a token that identifies the Apple event objects specified by the `objectSpecifier` parameter. (Token is defined in `AEDisposeToken` (page 41).)

Your object accessor functions may need to create many tokens to resolve a single object specifier; this parameter contains only the final token that identifies the requested Apple event object.

Whenever the `AEResolve` function returns final token to your event handler as the result of the resolving an object specifier passed to `AEResolve`, your application must deallocate the memory used by the token. If your application uses complex tokens, it must dispose of the token by calling `AEDisposeToken` (page 41). If your application uses simple tokens, you can use either `AEDisposeToken` (page 41) or `AEDisposeDesc` (page 40). See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). The `AEResolve` function returns any result code returned by one of your application's object accessor functions or object callback functions. For example, an object accessor function can return `errAENoSuchObject` (–1728) when it can't find an Apple event object, or it can return more specific result codes. If any object accessor function or object callback function returns a result code other than `noErr` or `errAEEventNotHandled`, `AEResolve` immediately disposes of any existing tokens and returns. The result code it returns in this case is the result code returned by the object accessor function or the object callback function.

Discussion

If an Apple event parameter consists of an object specifier, your handler for the event typically calls the `AEResolve` function to begin the process of resolving the object specifier.

The `AEResolve` function resolves the object specifier passed in the `objectSpecifier` parameter with the help of your object accessor functions, described in "Object Accessor Callbacks" (page 139), and the object callback functions, described in "Object Callback Functions" (page 140).

For information on how to receive error information from the `AEResolve` function, see `OSLGetErrDescProcPtr` (page 157).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEObjects.h`

AEResumeTheCurrentEvent

Informs the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event.

```
OSErr AEResumeTheCurrentEvent (
   const AppleEvent *theAppleEvent,
   const AppleEvent *reply,
   AEEventHandlerUPP dispatcher,
   SRefCon handlerRefcon
);
```

Parameters

theAppleEvent

A pointer to the Apple event to resume handling for. See `AppleEvent` (page 175).

reply

A pointer to the default reply provided by the Apple Event Manager for the Apple event. See `AppleEvent` (page 175).

dispatcher

One of the following:

- a universal procedural pointer to a function that the Apple Event Manager calls to handle the resumed event, or
- the constant `kAEUseStandardDispatch`, which tells the Apple Event Manager to handle the resumed event with its standard dispatching mechanism, or
- the constant `kAENoDispatch`, which tells the Apple Event Manager the Apple event has been completely processed and doesn't need to be dispatched.

See the `handlerRefcon` parameter for more information.

The dispatch constants are described in "Resume Event Dispatch Constants" (page 219).

See `AEEventHandlerUPP` (page 171).

handlerRefcon

If the `dispatcher` parameter specifies a universal procedure pointer to your routine, the reference constant is passed to your handler. If you pass the value `kAEUseStandardDispatch` or `kAENoDispatch` for the `dispatcher` parameter, you must pass 0 for the `handlerRefcon` parameter.

If the value of `dispatcher` is `kAEUseStandardDispatch`, the Apple Event Manager ignores the `handlerRefcon` parameter and instead passes the reference constant stored in the Apple event dispatch table entry for the resumed Apple event.

If the value of `dispatcher` is any other value then it is a universal procedure pointer to an event handler, and the Apple Event Manager passes the value from the `handlerRefcon` parameter as the reference constant when it calls the handler.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). This is the error result from the Apple event handler (or `errAEHandlerNotFound`). In most cases your application should ignore this error because it will be seen by the Apple event sender as the `keyErrorNumber` parameter in the reply.

Discussion

Applications call `AESuspendTheCurrentEvent` (page 113) to suspend handling of an Apple event and `AEResumeTheCurrentEvent` to resume it again. You typically call the `AESuspendTheCurrentEvent` function when your application needs to do some lengthy processing before responding to the event.

When your application calls the `AEResumeTheCurrentEvent` function, the Apple Event Manager resumes handling the specified Apple event using the handler specified in the `dispatcher` parameter, if any. If `kAENoDispatch` is specified in the `dispatcher` parameter, `AEResumeTheCurrentEvent` simply informs the Apple Event Manager that the specified event has been handled.

Special Considerations

This function is not thread-safe and, along with `AESuspendTheCurrentEvent`, should be called only on the main thread.

When your application suspends an Apple event, it does not need to dispose of the Apple event or the reply Apple event passed to the handler that suspends the event, whether or not the application eventually resumes the event. However, if the application will later resume the event, the handler that suspends the event should save a copy of the underlying data storage for the Apple event and the reply event. When resuming the event, you pass those copies to `AEResumeTheCurrentEvent`, which uses the information they contain to identify the original event and reply. For related information, see `AESuspendTheCurrentEvent` (page 113).

Make sure all processing involving the event or the reply has been completed before your application calls `AEResumeTheCurrentEvent`. Do not call `AEResumeTheCurrentEvent` for an event that was not suspended.

An Apple event handler that suspends an event should not immediately call `AEResumeTheCurrentEvent`, because the handler will generate an error. Instead, the handler should just return after suspending the event.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESend

Sends the specified Apple event.

```
OSErr AESend (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   AESendMode sendMode,
   AESendPriority sendPriority,
   SInt32 timeOutInTicks,
   AEIdleUPP idleProc,
   AEFilterUPP filterProc
);
```

Parameters

theAppleEvent

A pointer to the Apple event to send. See `AppleEvent` (page 175).

reply

A pointer to a reply Apple event. On return, contains the reply Apple event from the server application, if you specified the `kAEWaitReply` flag in the `sendMode` parameter. If you specify the `kAEQueueReply` flag in the `sendMode` parameter, you receive the reply Apple event in your event queue. If you specify `kAENoReply` flag, the reply Apple event is a null descriptor (one with descriptor type `typeNull`). If you specify `kAEWaitReply` in the `sendMode` parameter, and if the function returns successfully (see function result below), your application is responsible for using the `AEDisposeDesc` (page 40) function to dispose of the descriptor returned in the `reply` parameter.

sendMode

Specifies various options for how the server application should handle the Apple event. To obtain a value for this parameter, you add together constants to set bits that specify the reply mode, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. For more information, see "`AESendMode`" (page 182).

sendPriority

See the Version Notes section below for important information. A value that specifies the priority for processing the Apple event. You can specify normal or high priority, using the constants described in "`AESendMode`" (page 182). See `AESendPriority` (page 174).

timeOutInTicks

If the reply mode specified in the `sendMode` parameter is `kAEWaitReply`, or if a return receipt is requested, this parameter specifies the length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before timing out. Most applications should use the `kAEDefaultTimeout` constant, which tells the Apple Event Manager to provide an appropriate timeout duration. If the value of this parameter is `kNoTimeOut`, the Apple event never times out. These constants are described in "Timeout Constants" (page 221).

idleProc

A universal procedure pointer to a function that handles events (such as update, operating-system, activate, and null events) that your application receives while waiting for a reply. Your idle function can also perform other tasks (such as displaying a wristwatch or spinning beach ball cursor) while waiting for a reply or a return receipt.

If your application specifies the `kAEWaitReply` flag in the `sendMode` parameter and you wish your application to get periodic time while waiting for the reply to return, you must provide an idle function. Otherwise, you can pass a value of `NULL` for this parameter. For more information on the idle function, see `AEIdleProcPtr` (page 147).

filterProc

A universal procedure pointer to a function that determines which incoming Apple events should be received while the handler waits for a reply or a return receipt. If your application doesn't need to filter Apple events, you can pass a value of `NULL` for this parameter. If you do so, no application-oriented Apple events are processed while waiting. For more information on the filter function, see `AEFilterProcPtr` (page 146).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). The `AESend` function returns `noErr` if the Event Manager successfully sends the Apple event—this value does not indicate that the Apple event was handled successfully. If the handler returns a result code other than `noErr`, and if the client is waiting for a reply, `AESend` returns the result code in the `keyErrorNumber` parameter of the reply Apple event. For a result code other than `noErr`, you should not call the `AEDisposeDesc` (page 40) function to dispose of the descriptor returned in the `reply` parameter, because the descriptor is invalid.

Discussion

You typically create an Apple event to send with the `AECreateAppleEvent` (page 32) function and add information to it with the functions described in "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 14).

If the Apple Event Manager cannot find a handler for the Apple event in the server application's dispatch table or in the system dispatch table, it returns the result code `errAEEventNotHandled` to the server application (as the result of the `AEProcessAppleEvent` (page 73) function). If the client application is waiting for a reply, the Apple Event Manager also returns this result code to the client in the `keyErrorNumber` parameter of the reply event.

In addition to specifying the wait duration for replies, the `timeOutInTicks` parameter is used as a wait value when queuing events for other applications. The Apple Event Manager waits for the specified duration as it attempts to queue the event. If you specify `kAEWaitReply` and the target application quits or crashes after the event is queued but before the reply is returned, the Apple Event Manager returns a `sessionClosedErr` result code.

In some situations, there are advantages to sending Apple events with the `AESendMessage` (page 94) function. That function requires less overhead than `AESend` and it allows you to send Apple events without linking to the entire Carbon framework (and window server), as required by `AESend`. For more information on sending Apple events, see "Sending an Apple Event" in Apple Events Programming Guide.

Version Notes
In Mac OS 9 and earlier, you use the `sendMode` parameter to specify how the server should handle the Apple event. "`AESendMode`" (page 182) provides a complete description of the constants you use with this parameter. The `sendPriority` parameter is deprecated in Mac OS X and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESendMessage

Sends an AppleEvent to a target process without some of the overhead required by `AESend`.

```
OSStatus AESendMessage (
   const AppleEvent *event,
   AppleEvent *reply,
   AESendMode sendMode,
   long timeOutInTicks
);
```

Parameters

event

A pointer to the Apple event to send.

reply

A pointer to a reply Apple event. On return, contains the reply Apple event from the server application, if you specified the `kAEWaitReply` flag in the `sendMode` parameter. If you specify the `kAEQueueReply` flag in the `sendMode` parameter, you receive the reply Apple event in your event queue. If you specify `kAENoReply` flag, the reply Apple event is a null descriptor (one with descriptor type `typeNull`). If you specify `kAEWaitReply` in the `sendMode` parameter, and if the function returns successfully (see function result below), your application is responsible for using the `AEDisposeDesc` (page 40) function to dispose of the descriptor returned in the `reply` parameter.

sendMode

Specifies various options for how the server application should handle the Apple event. To obtain a value for this parameter, you add together constants to set bits that specify the reply mode, the interaction level, the application switch mode, the reconnection mode, and the return receipt mode. For more information, see "`AESendMode`" (page 182).

timeOutInTicks

If the reply mode specified in the `sendMode` parameter is `kAEWaitReply`, or if a return receipt is requested, this parameter specifies the length of time (in ticks) that the client application is willing to wait for the reply or return receipt from the server application before timing out. Most applications should use the `kAEDefaultTimeout` constant, which tells the Apple Event Manager to provide an appropriate timeout duration. If the value of this parameter is `kNoTimeOut`, the Apple event never times out. These constants are described in "Timeout Constants" (page 221).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
The `AESendMessage` function allows you to send Apple events without linking to the entire Carbon framework, as required by `AESend` (page 92). Linking with Carbon brings in the HIToolbox framework, which requires that your application have a connection to the window server. Daemons and other applications that have

no interface but wish to send and receive Apple events can use the following functions for working with Apple events at a lower level: `AESendMessage`, `AEGetRegisteredMachPort` (page 61), `AEDecodeMessage` (page 37), and `AEProcessMessage` (page 74). See the descriptions for those functions for more information on when you might use them.

If the target of an event sent with `AESendMessage` is the current process (as specified by using `typeProcessSerialNumber` of `{ 0, kCurrentProcess }` in the Apple event being sent), the Apple event is dispatched directly to the appropriate event handler in your process and not serialized.

Special Considerations

The `AESendMessage` function is both asynchronous and thread-safe, so you could, for example, set up a thread to send an Apple event and wait for a reply. If you use threads, you must add a `typeReplyPortAttr` attribute to your event that identifies the Mach port on which to receive the reply.

However, due to a bug that was present prior to Mac OS X version 10.5, you could not safely dispose of a Mach port you created to use as the reply port. Disposing of the port could, rarely, lead to a crash, while failing to dispose of if leaked resources. The sample code project *AESendThreadSafe* shows how to safely work around the bug in earlier Mac OS versions.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEMach.h`

AESetInteractionAllowed

Specifies user interaction preferences for responding to an Apple event when your application is the server application.

```
OSErr AESetInteractionAllowed (
   AEInteractAllowed level
);
```

Parameters

level

The desired user interaction level. Pass one of the values described in "User Interaction Level Constants" (page 221).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
If you don't set the user interaction level by calling `AESetInteractionAllowed`, the default level is `kAEInteractWithLocal` (which indicates that your server application may interact with the user in response to an Apple event only if the client application is on the same computer as the server application).

For additional information on interaction level, see `AESend` (page 92) and "`AESendMode`" (page 182).

See also `AESetInteractionAllowed` (page 95) and `AEInteractWithUser` (page 69).

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEInteraction.h

AESetObjectCallbacks

Specifies the object callback functions for your application.

```
OSErr AESetObjectCallbacks (
   OSLCompareUPP myCompareProc,
   OSLCountUPP myCountProc,
   OSLDisposeTokenUPP myDisposeTokenProc,
   OSLGetMarkTokenUPP myGetMarkTokenProc,
   OSLMarkUPP myMarkProc,
   OSLAdjustMarksUPP myAdjustMarksProc,
   OSLGetErrDescUPP myGetErrDescProcPtr
);
```

Parameters

myCompareProc

Either a universal procedure pointer to the object comparison function provided by your application or NULL if no function is provided. See OSLCompareUPP (page 177).

myCountProc

Either a universal procedure pointer to the object-counting function provided by your application or NULL if no function is provided. See OSLCountUPP (page 177).

myDisposeTokenProc

Either a universal procedure pointer to the token disposal function provided by your application or NULL if no function is provided. (Token is defined in AEDisposeToken (page 41). See OSLDisposeTokenUPP (page 177).

myGetMarkTokenProc

Either a universal procedure pointer to the function for returning a mark token provided by your application or NULL if no function is provided. See OSLGetMarkTokenUPP (page 178).

myMarkProc

Either a universal procedure pointer to the object-marking function provided by your application or NULL if no function is provided. See OSLMarkUPP (page 178).

myAdjustMarksProc

Either a universal procedure pointer to the mark-adjusting function provided by your application or NULL if no function is provided. See OSLAdjustMarksUPP (page 177).

myGetErrDescProcPtr

Either a universal procedure pointer to the error callback function provided by your application or NULL if no function is provided. See OSLGetErrDescUPP (page 178).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function is just a convenient wrapper for AEInstallSpecialHandler (page 68). You can manipulate the special handler table with more control using the routines described in "Managing Special Handler Dispatch Tables" (page 20).

Your application can provide only one each of the object callback functions specified by AESetObjectCallbacks—one object comparison function, one object-counting function, and so on. As a result, each of these callback functions must perform the requested task (comparing, counting, and so on)

for all the object classes that your application supports. In contrast, your application may provide many different object accessor functions if necessary, depending on the object classes and token types your application supports. You install object accessor functions with `AEInstallObjectAccessor` (page 67).

To replace object callback functions that have been previously installed, you can call `AESetObjectCallbacks` again. Each additional call to `AESetObjectCallbacks` replaces any object callback functions installed by previous calls. Only those functions you specify are replaced; to avoid replacing existing callback functions, specify a value of `NULL` for the functions you don't want to replace.

You cannot use `AESetObjectCallbacks` to replace system object callback functions or object accessor functions. To install system object callback functions, use the function `AEInstallSpecialHandler` (page 68).

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AESetTheCurrentEvent

Specifies a current Apple event to take the place of the one your application has suspended.

```
OSErr AESetTheCurrentEvent (
   const AppleEvent *theAppleEvent
);
```

Parameters

theAppleEvent
A pointer to the Apple event to handle as the current event. See `AppleEvent` (page 175).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
There is usually no reason for your application to use the `AESetTheCurrentEvent` function. Instead of calling this function, your application should let the Apple Event Manager set the current Apple event through its standard dispatch mechanism.

If you need to avoid the dispatch mechanism, you must use the `AESetTheCurrentEvent` function only in the following way:

1. Your application suspends handling of an Apple event by calling the `AESuspendTheCurrentEvent` (page 113) function.

2. Your application calls the `AESetTheCurrentEvent` function. This informs the Apple Event Manager that your application is handling the suspended Apple event. In this way, any functions that call the `AEGetTheCurrentEvent` (page 63) function can ascertain which event is currently being handled.

3. When your application finishes handling the Apple event, it calls the `AEResumeTheCurrentEvent` (page 90) function with the value `kAENoDispatch` to tell the Apple Event Manager that the event has been processed and need not be dispatched.

Special Considerations

This function is not thread-safe and should only be called on the main thread.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AESizeOfAttribute

Gets the size and descriptor type of an Apple event attribute from a descriptor of type `AppleEvent`.

```
OSErr AESizeOfAttribute (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAppleEvent
A pointer to the Apple event to get the attribute data from. See `AppleEvent` (page 175).

theAEKeyword
The keyword that specifies the attribute. Some keyword constants are described in "Keyword Attribute Constants" (page 209). See `AEKeyword` (page 172).

typeCode
A pointer to a descriptor type. On return, specifies the descriptor type of the attribute. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). Can be `NULL`. See `DescType` (page 176).

dataSize
A pointer to a size variable. On return, the length, in bytes, of the data in the attribute. Can be `NULL`.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfFlattenedDesc

Returns the amount of buffer space needed to store the descriptor after flattening it.

```
Size AESizeOfFlattenedDesc (
   const AEDesc *theAEDesc
);
```

Parameters

theAEDesc

A pointer to the descriptor to be flattened. See `AEDesc` (page 162).

Return Value

The size, in bytes, required to store the flattened descriptor.

Discussion

You call this function before calling `AEFlattenDesc` (page 42) to determine the required size of the buffer for the flatten operation.

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AESizeOfKeyDesc

Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord`.

```
OSErr AESizeOfKeyDesc (
    const AppleEvent *theAERecord,
    AEKeyword theAEKeyword,
    DescType *typeCode,
    Size *dataSize
);
```

Parameters

theAERecord

A pointer to the Apple event record to get the parameter data from.

theAEKeyword

The keyword that specifies the desired parameter. Some keyword parameter constants are described in "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataSize

A pointer to a size variable. On return, the length, in bytes, of the data in the Apple event parameter.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This function is declared as a macro that invokes `AESizeOfParam` (page 101), which can operate on an Apple event or an Apple event record. See the Discussion for that function for more information.

Version Notes
See `AESizeOfParam` (page 101).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfNthItem

Gets the data size and descriptor type of the descriptor at a specified position in a descriptor list.

```
OSErr AESizeOfNthItem (
   const AEDescList *theAEDescList,
   long index,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAEDescList
A pointer to the descriptor list containing the descriptor. See `AEDescList` (page 169).

index
A one-based positive integer indicating the position of the descriptor to get the data size for. `AESizeOfNthItem` returns an error if you pass zero, a negative number, or a value that is out of range.

typeCode
A pointer to a descriptor type. On return, specifies the descriptor type of the descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataSize
A pointer to a size variable. On return, the length (in bytes) of the data in the descriptor.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESizeOfParam

Gets the size and descriptor type of an Apple event parameter from a descriptor of type `AERecord` or `AppleEvent`.

```
OSErr AESizeOfParam (
   const AppleEvent *theAppleEvent,
   AEKeyword theAEKeyword,
   DescType *typeCode,
   Size *dataSize
);
```

Parameters

theAppleEvent

A pointer to the Apple event to get the parameter data from. See `AppleEvent` (page 175).

theAEKeyword

The keyword that specifies the desired parameter. Some keyword parameter constants are described in "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

typeCode

A pointer to a descriptor type. On return, specifies the descriptor type of the Apple event parameter. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataSize

A pointer to a size variable. On return, the length, in bytes, of the data in the Apple event parameter.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Version Notes

Thread safe starting in Mac OS X v10.2.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEDataModel.h`

AEStreamClose

Closes and deallocates an `AEStreamRef`.

```
OSStatus AEStreamClose (
   AEStreamRef ref,
   AEDesc *desc
);
```

Parameters

ref

An `AEStreamRef` (page 174)containing the stream data.

desc

A pointer to a descriptor for receiving a the stream data, or `NULL` if you want to discard the data. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this function to dispose of an `AEStreamRef` you created using `AEStreamCreateEvent` (page 103), `AEStreamOpen` (page 105), or `AEStreamOpenEvent` (page 106). To retrieve the resulting descriptor from the stream prior to disposal, pass in a pointer to an `AEDesc` structure in the *`desc`* parameter. If this parameter exists, `AEStreamClose` fills in the descriptor with the stream data. If the stream contains invalid information, possibly due to improperly balanced calls to "AEStream" functions, the returned descriptor type is set to `typeNull`.

Regardless of any errors returned by this function, it is always safe to call `AEDisposeDesc` (page 40) on the returned descriptor.

Specifying `NULL` for the *`desc`* parameter causes `AEStreamClose` to discard the stream data and dispose of the `AEStreamRef`. When you call `AEStreamClose` in this way, you do not need to worry about balancing nested calls to "AEStream" functions. This technique is particularly useful during error-handling situations where you need to dispose of a stream but do not know its exact state.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseDesc

Marks the end of a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamCloseDesc (
   AEStreamRef ref
);
```

Parameters

`ref`

An `AEStreamRef` (page 174)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Call this function to balance a preceding call to `AEStreamOpenDesc` (page 105) or `AEStreamOpenKeyDesc` (page 106). This function completes the definition of the `AEDesc`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseList

Marks the end of a list of descriptors in an `AEStreamRef`.

```
OSStatus AEStreamCloseList (
   AEStreamRef ref
);
```

Parameters

ref

An `AEStreamRef` (page 174)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Call this function to balance a preceding call to `AEStreamOpenList` (page 107). This function completes the definition of the `AEDescList`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCloseRecord

Marks the end of a record in an `AEStreamRef`.

```
OSStatus AEStreamCloseRecord (
   AEStreamRef ref
);
```

Parameters

ref

An `AEStreamRef` (page 174)containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Call this function to balance a preceding call to `AEStreamOpenRecord` (page 107). This function completes the definition of the Apple event record.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamCreateEvent

Creates a new Apple event and opens a stream for writing data to it.

```
AEStreamRef AEStreamCreateEvent (
   AEEventClass clazz,
   AEEventID id,
   DescType targetType,
   const void *targetData,
   Size targetLength,
   SInt16 returnID,
   SInt32 transactionID
);
```

Parameters

clazz

The event class of the Apple event. See `AEEventClass` (page 171).

id

The event ID of the Apple event. See `AEEventID` (page 172).

targetType

The address type for the addressing information in the next two parameters. Usually contains one of the following values: `typeApplSignature`. `typeKernelProcessID`, or `typeProcessSerialNumber`. See `DescType` (page 176).

targetData

A pointer to the address information. The data in this pointer must match the data associated with the specified *targetType*.

targetLength

The number of bytes pointed to by the `targetData` parameter.

returnID

The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID. The return ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 205). See `AEReturnID` (page 174).

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events. This transaction ID constant is described in "ID Constants for the AECreateAppleEvent Function" (page 205). See `AETransactionID` (page 175).

Return Value

An `AEStreamRef` (page 174) associated with the new event.

Discussion

This routine effectively combines a call to `AECreateAppleEvent` (page 32) followed by a call to `AEStreamOpenEvent` (page 106) to create a new Apple event in the stream. You can use the returned `AEStreamRef` to add parameters to the new Apple event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

AEStreamOpen

Opens a new `AEStreamRef` for use in building a descriptor.

```
AEStreamRef AEStreamOpen (
   void
);
```

Return Value
A new `AEStreamRef` (page 174) or `NULL` if the stream data structures cannot be allocated.

Discussion
This function creates a new stream for use in describing the contents of a descriptor, descriptor list, or Apple event record (`AEDesc`, `AEDescList`, or `AERecord`).

You can use the returned `AEStreamRef` with other "AEStream" routines to build the contents of a descriptor. When you are done building the descriptor, use `AEStreamClose` (page 101) to close the stream.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenDesc

Marks the beginning of a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamOpenDesc (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref
An `AEStreamRef` (page 174) containing the stream data.

newType
A type code for the new `AEDesc` being added to the stream. See `DescType` (page 176).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this routine to mark the beginning of a descriptor definition in an `AEDesc`. After calling this routine, you should call `AEStreamWriteData` (page 110) one or more times to write the descriptor data to the stream. When you are done writing data, you must call `AEStreamCloseDesc` (page 102) to complete the descriptor definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenEvent

Opens a stream for an existing Apple event.

```
AEStreamRef AEStreamOpenEvent (
   AppleEvent *event
);
```

Parameters

event

An existing Apple event. See `AppleEvent` (page 175).

Return Value

An `AEStreamRef` (page 174) for the Apple event or `NULL` if the stream data structures could not be allocated.

Discussion

Use this function to open a stream and add parameters to an existing Apple event. This function copies any parameters already in the Apple event to the stream prior to returning the `AEStreamRef`. When you are done adding parameters, use `AEStreamClose` (page 101) to save them to the Apple event and close the stream.

If there is not enough available storage to complete the operation, `AEStreamOpenEvent` returns `NULL` and leaves the Apple event unchanged.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

AEStreamOpenKeyDesc

Marks the beginning of a key descriptor in an `AEStreamRef`.

```
OSStatus AEStreamOpenKeyDesc (
   AEStreamRef ref,
   AEKeyword key,
   DescType newType
);
```

Parameters

ref

An `AEStreamRef` (page 174) containing the stream data.

key

The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 172).

newType

A type code for the new `AEDesc` being added to the stream. See `DescType` (page 176).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this routine to mark the beginning of a keyword/descriptor definition in an Apple event record. After calling this routine, you should call `AEStreamWriteData` (page 110) one or more times to write the record data to the stream. When you are done writing data, you must call `AEStreamCloseDesc` (page 102) to complete the record definition.

This routine must be called only as part of an Apple event record definition. You cannot use this routine to write keyword/descriptor definitions to other descriptor types, such as an `AEDesc` or `AEDescList`, even if those types are nested inside an Apple event record. In situations where you need to create nested records, this routine opens a new keyword/descriptor definition in the Apple event record associated with the most recent call to `AEStreamOpenRecord` (page 107).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenList

Marks the beginning of a descriptor list in an `AEStreamRef`.

```
OSStatus AEStreamOpenList (
   AEStreamRef ref
);
```

Parameters
ref
An `AEStreamRef` (page 174) containing the stream data.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
This routine marks the beginning of a sequence of zero or more descriptor definitions that you use to build an `AEDescList` structure. After calling this routine, you can write any number of `AEDesc`, `AEDescList`, or `AERecord` structures to the stream as elements of the list. When you are done, you must call `AEStreamCloseList` (page 102) to complete the `AEDescList` definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamOpenRecord

Marks the beginning of an Apple event record in an `AEStreamRef`.

```
OSStatus AEStreamOpenRecord (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref

An `AEStreamRef` (page 174) containing the stream data.

newType

A type code for the new record you are adding to the stream. This value can be `typeAERecord` or any other appropriate value. See `DescType` (page 176).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

This routine marks the beginning of a sequence of zero or more keyword/descriptor definitions that you use to build an `AERecord` structure. You must balance each call to this method with a corresponding call to `AEStreamCloseRecord` (page 103).

For information on adding keyword/descriptor data to the record, see the `AEStreamOpenKeyDesc` (page 106), `AEStreamWriteKey` (page 111), and `AEStreamWriteKeyDesc` (page 112) routines.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

AEStreamOptionalParam

Designates a parameter in an Apple event as optional.

```
OSStatus AEStreamOptionalParam (
   AEStreamRef ref,
   AEKeyword key
);
```

Parameters

ref

An `AEStreamRef` (page 174) containing the stream data.

key

The `AEKeyword` associated with any keyword/descriptor pair in an Apple event. See `AEKeyword` (page 172).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

Calls to this routine must be preceded by a call to either `AEStreamCreateEvent` (page 103) or `AEStreamOpenEvent` (page 106).

The descriptor associated with the specified *key* does not need to exist before you call this routine.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamSetRecordType

Sets the type of the most recently created record in an `AEStreamRef`.

```
OSStatus AEStreamSetRecordType (
   AEStreamRef ref,
   DescType newType
);
```

Parameters

ref
An `AEStreamRef` (page 174) containing the stream data.

newType
The new type code for the `AERecord` being added to the stream. See `DescType` (page 176).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this routine to change the type of a record after it has been opened. You must call this routine between calls to `AEStreamOpenRecord` (page 107) and `AEStreamCloseRecord` (page 103). The type you specify in the *newType* parameter replaces the previous type specified by `AEStreamOpenRecord` (page 107).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteAEDesc

Copies an existing descriptor into an `AEStreamRef`.

```
OSStatus AEStreamWriteAEDesc (
   AEStreamRef ref,
   const AEDesc *desc
);
```

Parameters

ref
An `AEStreamRef` (page 174) containing the stream data.

desc
A pointer to the descriptor you want to copy into the stream. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You can use this routine to incorporate an existing descriptor into the stream. For example, you could use this routine if you had a complex descriptor you wanted to add to multiple streams, but which would be costly to create each time.

Do not use `AEStreamOpenDesc` (page 105) and `AEStreamCloseDesc` (page 102) with this routine to open and close the descriptor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteData

Appends data to the current descriptor in an `AEStreamRef`.

```
OSStatus AEStreamWriteData (
   AEStreamRef ref,
   const void *data,
   Size length
);
```

Parameters

ref
An `AEStreamRef` (page 174) containing the stream data.

data
A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length
The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You can call this routine any number of times to build up the data contents of the descriptor incrementally. You must precede calls to this routine by a call to either `AEStreamOpenDesc` (page 105) or `AEStreamOpenKeyDesc` (page 106). When you are done adding data to the descriptor, call `AEStreamCloseDesc` (page 102) to complete the descriptor definition.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteDesc

Appends the data for a complete descriptor to an `AEStreamRef`.

```
OSStatus AEStreamWriteDesc (
   AEStreamRef ref,
   DescType newType,
   const void *data,
   Size length
);
```

Parameters

ref

An `AEStreamRef` (page 174) containing the stream data.

newType

A type code for the new `AEDesc` being added to the stream. See `DescType` (page 176).

data

A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length

The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this routine to write the data for a descriptor to the stream. When using this routine, you must supply all of the descriptor data at once.

Do not use `AEStreamOpenDesc` (page 105) and `AEStreamCloseDesc` (page 102) with this routine to open and close the descriptor.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteKey

Marks the beginning of a keyword/descriptor pair for a descriptor in an `AEStreamRef`.

```
OSStatus AEStreamWriteKey (
   AEStreamRef ref,
   AEKeyword key
);
```

Parameters

ref

An `AEStreamRef` (page 174) containing the stream data.

key

The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 172).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You must follow this call with a sequence of "AEStream" calls to specify exactly one descriptor that goes with the keyword. The descriptor you create can be of type `AEDesc`, `AEDescList`, or `AERecord`.

If you are creating nested descriptors, this routine begins a new keyword/descriptor pair for the descriptor most recently opened by a call to `AEStreamWriteKey` (page 111) or `AEStreamOpenEvent` (page 106). You cannot use this routine to write parameters to any other types of descriptors, even if they are nested inside of an `AERecord`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AEStreamWriteKeyDesc

Writes a complete keyword/descriptor pair to an `AEStreamRef`.

```
OSStatus AEStreamWriteKeyDesc (
   AEStreamRef ref,
   AEKeyword key,
   DescType newType,
   const void *data,
   Size length
);
```

Parameters

ref
An `AEStreamRef` (page 174) containing the stream data.

key
The `AEKeyword` associated with the new descriptor being added to the stream. See `AEKeyword` (page 172).

newType
A type code for the new `AEDesc` being added to the stream. See `DescType` (page 176).

data
A pointer to the block of memory containing the descriptor data. This routine copies the memory block immediately, so you do not need to retain it for the benefit of this routine.

length
The number of bytes pointed to by the *data* parameter.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Use this routine to add a descriptor to the currently open `AERecord` inside a stream. You cannot use this routine to write parameters to any other types of descriptors, even if they are nested inside of an `AERecord`. This routine can only be called in between calls to `AEStreamOpenRecord` (page 107) and `AEStreamCloseRecord` (page 103).

This method is analogous to the Apple Event Manager routine `AEPutParamPtr` (page 80), except it is for use with streams.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AESuspendTheCurrentEvent

Suspends the processing of the Apple event that is currently being handled.

```
OSErr AESuspendTheCurrentEvent (
   const AppleEvent *theAppleEvent
);
```

Parameters

theAppleEvent

A pointer to the Apple event to suspend handling for. If the pointed-to Apple event is not the current event, AESuspendTheCurrentEvent does nothing and returns noErr. See AppleEvent (page 175).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
You typically call the AESuspendTheCurrentEvent function from an Apple event handler function, such as when your application needs to do some lengthy processing before responding to the event. After a successful call to this function, you are not required to return a result or a reply for the Apple event that was being handled. You can, however, return a result if you later call the AEResumeTheCurrentEvent (page 90) function to resume event processing.

Whether you will resume the suspended Apple event or not, you do not need to dispose of the Apple event or the reply Apple event passed to your handler. However, if your handler will later resume the event, you must save a copy of the underlying data storage for the Apple event and the reply event. When resuming the event, you pass those copies to AEResumeTheCurrentEvent (page 90), which uses the information they contain to identify the original event and reply.

You cannot merely save the pointers that are passed to your handler because they do not persist after your handler returns (although the underlying Apple events do persist). Use a function such as AEDuplicateDesc (page 42) to obtain copies of the Apple event and reply event. Later, after calling AEResumeTheCurrentEvent to resume the event, call AEDisposeDesc (page 40) to dispose of the copies.

Special Considerations

This function is not thread-safe and, along with AEResumeTheCurrentEvent, should be called only on the main thread.

If your application suspends handling of an Apple event it sends to itself, the Apple Event Manager immediately returns from the AESend (page 92) call with the error code errAETimeout, regardless of the parameters specified in the call to AESend. The function calling AESend should take the timeout error as confirmation that the event was sent.

As with other calls to AESend that return a timeout error, the handler continues to process the event nevertheless. The handler's reply, if any, is provided in the reply event when the handling is completed. The Apple Event Manager provides no notification that the reply is ready. If no data has yet been placed in the reply event, the Apple Event Manager returns errAEReplyNotArrived when your application attempts to extract data from the reply.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEUnflattenDesc

Unflattens the data in the passed buffer and creates a descriptor from it.

```
OSStatus AEUnflattenDesc (
   const void *buffer,
   AEDesc *result
);
```

Parameters

buffer

A pointer to memory, allocated by the application, that contains flattened data produced by a previous call to `AEFlattenDesc` (page 42).

result

A null descriptor. On successful completion, points to a descriptor created from the flattened data. The caller is responsible for disposing of the descriptor.

Return Value
A result code. Returns `paramErr` if the flattened data in `buffer` is found to be invalid. See "Apple Event Manager Result Codes" (page 252) for other possible values.

Discussion
This function assumes the passed buffer contains valid flattened data, produced by a previous call to `AEFlattenDesc` (page 42). See that function for a description of when you might want to flatten and unflatten descriptors, and of possible limitations.

Flattening and unflattening works across OS versions, including between Mac OS 9 and Mac OS X.

Flattening is endian-neutral. That is, you can save flattened data on a machine that is either big-endian or little-endian, then retrieve and unflatten the data on either type of machine, without any special steps by your application.

Version Notes
Thread safe starting in Mac OS X v10.2.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

CreateCompDescriptor

Creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

```
OSErr CreateCompDescriptor (
   DescType comparisonOperator,
   AEDesc *operand1,
   AEDesc *operand2,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

comparisonOperator

The comparison operator for comparing the descriptors in the `operand1` and `operand2` parameters. The standard comparison operators are defined in "Comparison Operator Constants" (page 190).

The actual comparison of the two operands is performed by the object comparison function provided by the client application. The way a comparison operator is interpreted is up to each application.

See `DescType` (page 176).

operand1

A pointer to an object specifier. See `AEDesc` (page 162).

operand2

A pointer to a descriptor (which can be an object specifier or any other descriptor) whose value is compared to the value of `operand1`. See `AEDesc` (page 162).

disposeInputs

A Boolean value. Pass `TRUE` if the function should automatically dispose of any descriptors you have provided in the `operand1` and `operand2` parameters to the function. Pass `FALSE` if your application will dispose of the descriptors itself. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the comparison descriptor created by `CreateCompDescriptor`. Your application must dispose of this descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEPackObject.h`

CreateLogicalDescriptor

Creates a logical descriptor that specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

```
OSErr CreateLogicalDescriptor (
   AEDescList *theLogicalTerms,
   DescType theLogicOperator,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

theLogicalTerms

A pointer to a list containing comparison descriptors (typeLogicalDescriptor), logical descriptors (typeCompDescriptor), or both. If the value of the parameter theLogicOperator is kAEAND or kAEOR, the list can contain any number of descriptors. If the value of the parameter theLogicOperator is kAENOT, logically this list should contain a single descriptor. However, the function will not return an error if the list contains more than one descriptor for a logical operator of kAENOT. See AEDescList (page 169).

theLogicOperator

A logical operator represented by one of the constants described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191). What you pass for this parameter helps determine what you pass for the theLogicalTerms parameter. See DescType (page 176).

disposeInputs

A Boolean value. Pass TRUE if the function should automatically dispose of the descriptors you have provided in the theLogicalTerms parameter or (FALSE) if your application will. A value of FALSE may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the logical descriptor created by CreateLogicalDescriptor. Your application must dispose of this descriptor after it has finished using it. See AEDesc (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion

The CreateLogicalDescriptor function creates a logical descriptor, which specifies a logical operator and one or more logical terms for the Apple Event Manager to evaluate.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEPackObject.h

CreateObjSpecifier

Assembles an object specifier that identifies one or more Apple event objects, from other descriptors.

```
OSErr CreateObjSpecifier (
   DescType desiredClass,
   AEDesc *theContainer,
   DescType keyForm,
   AEDesc *keyData,
   Boolean disposeInputs,
   AEDesc *objSpecifier
);
```

Parameters

desiredClass

The object class of the desired Apple event objects. See `DescType` (page 176).

theContainer

A pointer to a descriptor that describes the container for the requested object, usually in the form of another object specifier. See `AEDesc` (page 162).

keyForm

The key form for the object specifier.

keyData

A pointer to a descriptor that supplies the key data for the object specifier.

disposeInputs

A Boolean value. Pass (`TRUE`) if the function should dispose of the descriptors for the `theContainer` and `keyData` parameters or (`FALSE`) if your application will. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

objSpecifier

On successful return, a pointer to the object specifier created by the `CreateObjSpecifier` function. If the function returns successfully, your application should call the `AEDisposeDesc` (page 40) function to dispose of this descriptor after it has finished using it.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEPackObject.h`

CreateOffsetDescriptor

Creates an offset descriptor that specifies the position of an element in relation to the beginning or end of its container.

```
OSErr CreateOffsetDescriptor (
   long theOffset,
   AEDesc *theDescriptor
);
```

Parameters

theOffset

A positive integer that specifies the offset from the beginning of the container (the first element has an offset of 1), or a negative integer that specifies the offset from the end (the last element has an offset of –1).

theDescriptor

A pointer to a descriptor. On successful return, the offset descriptor created by `CreateOffsetDescriptor`. On error, returns a null descriptor. Your application must dispose of the descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEPackObject.h`

CreateRangeDescriptor

Creates a range descriptor that specifies a series of consecutive elements in the same container.

```
OSErr CreateRangeDescriptor (
   AEDesc *rangeStart,
   AEDesc *rangeStop,
   Boolean disposeInputs,
   AEDesc *theDescriptor
);
```

Parameters

rangeStart

A pointer to an object specifier that identifies the first Apple event object in the range. See `AEDesc` (page 162).

rangeStop

A pointer to an object specifier that identifies the last Apple event object in the range. See `AEDesc` (page 162).

disposeInputs

A Boolean value. Pass (`TRUE`) if the function should dispose of the descriptors for the `rangeStart` and `rangeStop` parameters and set them to the null descriptor or (`FALSE`) if your application will. A value of `FALSE` may be more efficient for some applications because it allows them to reuse descriptors.

theDescriptor

A pointer to a descriptor. On successful return, the range descriptor created by `CreateRangeDescriptor`. Your application must dispose of this descriptor after it has finished using it. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Although the `rangeStart` and `rangeStop` parameters can be any object specifiers—including object specifiers that specify more than one Apple event object—most applications expect these parameters to specify single Apple event objects.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEPackObject.h`

DisposeAECoerceDescUPP

Disposes of a universal procedure pointer to a function that coerces data stored in a descriptor.

```
void DisposeAECoerceDescUPP (
   AECoerceDescUPP userUPP
);
```

Discussion
See the `AECoerceDescProcPtr` (page 140) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAECoercePtrUPP

Disposes of a universal procedure pointer to a function that coerces data stored in a buffer.

```
void DisposeAECoercePtrUPP (
   AECoercePtrUPP userUPP
);
```

Discussion
See the `AECoercePtrProcPtr` (page 141) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAEDisposeExternalUPP

Disposes of a universal procedure pointer to a function that disposes of data supplied to the `AECreateDescFromExternalPtr` function.

```
void DisposeAEDisposeExternalUPP (
   AEDisposeExternalUPP userUPP
);
```

Parameters
userUPP
The universal procedure pointer to be disposed of. See `AEDisposeExternalUPP` (page 171).

Discussion
See the `AECreateDescFromExternalPtr` (page 34) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

DisposeAEEventHandlerUPP

Disposes of a universal procedure pointer to an event handler function.

```
void DisposeAEEventHandlerUPP (
   AEEventHandlerUPP userUPP
);
```

Discussion
See the `AEEventHandlerProcPtr` (page 144) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DisposeAEFilterUPP

Disposes of a universal procedure pointer to an Apple event filter function.

```
void DisposeAEFilterUPP (
   AEFilterUPP userUPP
);
```

Discussion
See the `AEFilterProcPtr` (page 146) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

DisposeAEIdleUPP

Disposes of a universal procedure pointer to an Apple event idle function.

```
void DisposeAEIdleUPP (
   AEIdleUPP userUPP
);
```

Discussion
See the `AEIdleProcPtr` (page 147) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

DisposeOSLAccessorUPP

Disposes of a universal procedure pointer to an object accessor function.

```
void DisposeOSLAccessorUPP (
   OSLAccessorUPP userUPP
);
```

Discussion
See the `OSLAccessorProcPtr` (page 149) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLAdjustMarksUPP

Disposes of a universal procedure pointer to an object callback adjust marks function.

```
void DisposeOSLAdjustMarksUPP (
   OSLAdjustMarksUPP userUPP
);
```

Discussion
See the `OSLAdjustMarksProcPtr` (page 151) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLCompareUPP

Disposes of a universal procedure pointer to an object callback comparison function.

```
void DisposeOSLCompareUPP (
   OSLCompareUPP userUPP
);
```

Discussion
See the `OSLCompareProcPtr` (page 152) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLCountUPP

Disposes of a universal procedure pointer to an object callback count function.

```
void DisposeOSLCountUPP (
   OSLCountUPP userUPP
);
```

Discussion
See the `OSLCountProcPtr` (page 154) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLDisposeTokenUPP

Disposes of a universal procedure pointer to an object callback dispose token function.

```
void DisposeOSLDisposeTokenUPP (
   OSLDisposeTokenUPP userUPP
);
```

Discussion
See the `OSLDisposeTokenProcPtr` (page 155) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLGetErrDescUPP

Disposes of a universal procedure pointer to an object callback get error descriptor function.

```
void DisposeOSLGetErrDescUPP (
   OSLGetErrDescUPP userUPP
);
```

Discussion
See the `OSLGetErrDescProcPtr` (page 157) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLGetMarkTokenUPP

Disposes of a universal procedure pointer to an object callback get mark function.

```
void DisposeOSLGetMarkTokenUPP (
   OSLGetMarkTokenUPP userUPP
);
```

Discussion
See the `OSLGetMarkTokenProcPtr` (page 158) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

DisposeOSLMarkUPP

Disposes of a universal procedure pointer to an object callback mark function.

```
void DisposeOSLMarkUPP (
   OSLMarkUPP userUPP
);
```

Discussion
See the `OSLMarkProcPtr` (page 160) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeAECoerceDescUPP

Calls a universal procedure pointer to a function that coerces data stored in a descriptor.

```
OSErr InvokeAECoerceDescUPP (
   const AEDesc *fromDesc,
   DescType toType,
   SRefCon handlerRefcon,
   AEDesc *toDesc,
   AECoerceDescUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `AECoerceDescProcPtr` (page 140) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAECoercePtrUPP

Calls a universal procedure pointer to a function that coerces data stored in a buffer.

```
OSErr InvokeAECoercePtrUPP (
   DescType typeCode,
   const void *dataPtr,
   Size dataSize,
   DescType toType,
   SRefCon handlerRefcon,
   AEDesc *result,
   AECoercePtrUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `AECoercePtrProcPtr` (page 141) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAEDisposeExternalUPP

Calls a dispose external universal procedure pointer.

```
void InvokeAEDisposeExternalUPP (
   const void *dataPtr,
   Size dataLength,
   SRefCon refcon,
   AEDisposeExternalUPP userUPP
);
```

Parameters

dataPtr
: A pointer to the data to be disposed of. The data must be immutable and must not be freed until this UPP is called.

dataLength
: The length, in bytes, of the data to be disposed of.

refcon
: A reference constant, supplied by your application, that you can use in your dispose function.

Discussion
See the `AEDisposeExternalProcPtr` (page 143) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

InvokeAEEventHandlerUPP

Calls an event handler universal procedure pointer.

```
OSErr InvokeAEEventHandlerUPP (
   const AppleEvent *theAppleEvent,
   AppleEvent *reply,
   SRefCon handlerRefcon,
   AEEventHandlerUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `AEEventHandlerProcPtr` (page 144) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

InvokeAEFilterUPP

Calls an Apple event filter universal procedure pointer.

```
Boolean InvokeAEFilterUPP (
   EventRecord *theEvent,
   SInt32 returnID,
   AETransactionID transactionID,
   const AEAddressDesc *sender,
   AEFilterUPP userUPP
);
```

Return Value
The return value of the callback function. The filter routine returns `TRUE` to accept the Apple event or `FALSE` to filter it out.

Discussion
See the `AEFilterProcPtr` (page 146) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

InvokeAEIdleUPP

Calls an Apple event idle universal procedure pointer.

```
Boolean InvokeAEIdleUPP (
   EventRecord *theEvent,
   SInt32 *sleepTime,
   RgnHandle *mouseRgn,
   AEIdleUPP userUPP
);
```

Return Value
The return value of the callback function. The filter routine returns `TRUE` if your application is no longer willing to wait for a reply from the server or for the user to bring the application to the front. It returns `FALSE` if your application is still willing to wait.

Discussion
See the `AEIdleProcPtr` (page 147) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

InvokeOSLAccessorUPP

Calls an object accessor universal procedure pointer.

```
OSErr InvokeOSLAccessorUPP (
   DescType desiredClass,
   const AEDesc *container,
   DescType containerClass,
   DescType form,
   const AEDesc *selectionData,
   AEDesc *value,
   SRefCon accessorRefcon,
   OSLAccessorUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLAccessorProcPtr` (page 149) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLAdjustMarksUPP

Calls an object callback adjust marks universal procedure pointer.

```
OSErr InvokeOSLAdjustMarksUPP (
   long newStart,
   long newStop,
   const AEDesc *markToken,
   OSLAdjustMarksUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLAdjustMarksProcPtr` (page 151) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLCompareUPP

Calls an object callback comparison universal procedure pointer.

```
OSErr InvokeOSLCompareUPP (
   DescType oper,
   const AEDesc *obj1,
   const AEDesc *obj2,
   Boolean *result,
   OSLCompareUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLCompareProcPtr` (page 152) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLCountUPP

Calls an object callback count universal procedure pointer.

```
OSErr InvokeOSLCountUPP (
   DescType desiredType,
   DescType containerClass,
   const AEDesc *container,
   long *result,
   OSLCountUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLCountProcPtr` (page 154) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLDisposeTokenUPP

Calls an object callback dispose token universal procedure pointer.

```
OSErr InvokeOSLDisposeTokenUPP (
   AEDesc *unneededToken,
   OSLDisposeTokenUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLDisposeTokenProcPtr` (page 155) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

InvokeOSLGetErrDescUPP

Calls an object callback get error descriptor universal procedure pointer.

```
OSErr InvokeOSLGetErrDescUPP (
   AEDesc **appDescPtr,
   OSLGetErrDescUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLGetErrDescProcPtr` (page 157) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

InvokeOSLGetMarkTokenUPP

Calls an object callback get mark universal procedure pointer.

```
OSErr InvokeOSLGetMarkTokenUPP (
   const AEDesc *dContainerToken,
   DescType containerClass,
   AEDesc *result,
   OSLGetMarkTokenUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLGetMarkTokenProcPtr` (page 158) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

InvokeOSLMarkUPP

Calls an object callback mark universal procedure pointer.

```
OSErr InvokeOSLMarkUPP (
   const AEDesc *dToken,
   const AEDesc *markToken,
   long index,
   OSLMarkUPP userUPP
);
```

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
See the `OSLMarkProcPtr` (page 160) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewAECoerceDescUPP

Creates a new universal procedure pointer to a function that coerces data stored in a descriptor.

```
AECoerceDescUPP NewAECoerceDescUPP (
   AECoerceDescProcPtr userRoutine
);
```

Return Value
See AECoerceDescUPP (page 168).

Discussion
See the AECoerceDescProcPtr (page 140) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

NewAECoercePtrUPP

Creates a new universal procedure pointer to a function that coerces data stored in a buffer.

```
AECoercePtrUPP NewAECoercePtrUPP (
   AECoercePtrProcPtr userRoutine
);
```

Return Value
See AECoercePtrUPP (page 168).

Discussion
See the AECoercePtrProcPtr (page 141) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

NewAEDisposeExternalUPP

Creates a new universal procedure pointer to a function that disposes of data stored in a buffer.

```
AEDisposeExternalUPP NewAEDisposeExternalUPP (
   AEDisposeExternalProcPtr userRoutine
);
```

Return Value
See AEDisposeExternalUPP (page 171).

Discussion
See the AEDisposeExternalProcPtr (page 143) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

NewAEEventHandlerUPP

Creates a new universal procedure pointer to an event handler function.

```
AEEventHandlerUPP NewAEEventHandlerUPP (
   AEEventHandlerProcPtr userRoutine
);
```

Return Value
See `AEEventHandlerUPP` (page 171).

Discussion
See the `AEEventHandlerProcPtr` (page 144) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

NewAEFilterUPP

Creates a new universal procedure pointer to an Apple event filter function.

```
AEFilterUPP NewAEFilterUPP (
   AEFilterProcPtr userRoutine
);
```

Return Value
See `AEFilterUPP` (page 172).

Discussion
See the `AEFilterProcPtr` (page 146) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

NewAEIdleUPP

Creates a new universal procedure pointer to an Apple event idle function.

```
AEIdleUPP NewAEIdleUPP (
   AEIdleProcPtr userRoutine
);
```

Return Value
See `AEIdleUPP` (page 172).

Discussion
See the `AEIdleProcPtr` (page 147) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

NewOSLAccessorUPP

Creates a new universal procedure pointer to an object accessor function.

```
OSLAccessorUPP NewOSLAccessorUPP (
   OSLAccessorProcPtr userRoutine
);
```

Return Value
See `OSLAccessorUPP` (page 176).

Discussion
See the `OSLAccessorProcPtr` (page 149) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLAdjustMarksUPP

Creates a new universal procedure pointer to an object callback adjust marks function.

```
OSLAdjustMarksUPP NewOSLAdjustMarksUPP (
   OSLAdjustMarksProcPtr userRoutine
);
```

Return Value
See `OSLAdjustMarksUPP` (page 177).

Discussion
See the `OSLAdjustMarksProcPtr` (page 151) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLCompareUPP

Creates a new universal procedure pointer to an object callback comparison function.

```
OSLCompareUPP NewOSLCompareUPP (
   OSLCompareProcPtr userRoutine
);
```

Return Value
See `OSLCompareUPP` (page 177).

Discussion
See the `OSLCompareProcPtr` (page 152) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLCountUPP

Creates a new universal procedure pointer to an object callback count function.

```
OSLCountUPP NewOSLCountUPP (
   OSLCountProcPtr userRoutine
);
```

Return Value
See `OSLCountUPP` (page 177).

Discussion
See the `OSLCountProcPtr` (page 154) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

NewOSLDisposeTokenUPP

Creates a new universal procedure pointer to an object callback dispose token function.

```
OSLDisposeTokenUPP NewOSLDisposeTokenUPP (
   OSLDisposeTokenProcPtr userRoutine
);
```

Return Value
See `OSLDisposeTokenUPP` (page 177).

Discussion
See the `OSLDisposeTokenProcPtr` (page 155) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLGetErrDescUPP

Creates a new universal procedure pointer to an object callback get error descriptor function.

```
OSLGetErrDescUPP NewOSLGetErrDescUPP (
   OSLGetErrDescProcPtr userRoutine
);
```

Return Value
See OSLGetErrDescUPP (page 178).

Discussion
See the OSLGetErrDescProcPtr (page 157) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLGetMarkTokenUPP

Creates a new universal procedure pointer to an object callback get mark function.

```
OSLGetMarkTokenUPP NewOSLGetMarkTokenUPP (
   OSLGetMarkTokenProcPtr userRoutine
);
```

Return Value
See OSLGetMarkTokenUPP (page 178).

Discussion
See the OSLGetMarkTokenProcPtr (page 158) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

NewOSLMarkUPP

Creates a new universal procedure pointer to an object callback mark function.

```
OSLMarkUPP NewOSLMarkUPP (
   OSLMarkProcPtr userRoutine
);
```

Return Value
See `OSLMarkUPP` (page 178).

Discussion
See the `OSLMarkProcPtr` (page 160) callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

vAEBuildAppleEvent

Allows you to encapsulate calls to `AEBuildAppleEvent` in a wrapper routine.

```
OSStatus vAEBuildAppleEvent (
   AEEventClass theClass,
   AEEventID theID,
   DescType addressType,
   const void *addressData,
   Size addressLength,
   SInt16 returnID,
   SInt32 transactionID,
   AppleEvent *resultEvt,
   AEBuildError *error,
   const char *paramsFmt,
   va_list args
);
```

Parameters

`theClass`
The event class for the resulting Apple event. See `AEEventClass` (page 171).

`theID`
The event id for the resulting Apple event. See `AEEventID` (page 172).

`addressType`
The address type for the addressing information described in the next two parameters: usually one of `typeApplSignature`, `typeProcessSerialNumber`, or `typeKernelProcessID`. See `DescType` (page 176).

`addressData`
A pointer to the address information.

`addressLength`
The number of bytes pointed to by the `addressData` parameter.

`returnID`
The return ID for the created Apple event. If you pass a value of `kAutoGenerateReturnID`, the Apple Event Manager assigns the created Apple event a return ID that is unique to the current session. If you pass any other value, the Apple Event Manager assigns that value for the ID.

transactionID

The transaction ID for this Apple event. A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID. You can specify the `kAnyTransactionID` constant if the Apple event is not one of a series of interdependent Apple events.

result

A pointer to a descriptor where the resulting descriptor should be stored. See `AppleEvent` (page 175) for a description of the data type.

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See "AEBuild Error Codes" (page 179) for the syntax error codes that can be returned in this structure.

paramsFmt

An AEBuild format string describing the AppleEvent record to be created. The format of these strings is described in Technical Note TN2106, AEBuild*, AEPrint*, and Friends.

args

A variable array of arguments to be substituted into the `paramsFmt` format string. See the ANSI C Interfaces documentation for a description of the `va_list` data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Passing an argument list to `vAEBuildAppleEvent` corresponds to passing a series of individual parameters to the `AEBuildAppleEvent` (page 24) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `vAEBuildAppleEvent` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

vAEBuildDesc

Allows you to encapsulate calls to `AEBuildDesc` in your own wrapper routines.

```
OSStatus vAEBuildDesc (
   AEDesc *dst,
   AEBuildError *error,
   const char *src,
   va_list args
);
```

Parameters

dst

A pointer to a descriptor where the resulting descriptor should be stored. See `AEDesc` (page 162).

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 162).

src

An *`AEBuild`* format string describing the descriptor to be created.

args

A reference to a previously defined, variable argument parameter list to use with the descriptor-string. The file `<stdarg.h>` defines macros for declaring and using the `va_list` data type.

Return Value

A numeric result code indicating the success of the call. A value of `AEBuildSyntaxNoErr` (zero) means the call succeeded. You can use the `error` parameter to discover information about other errors. See "Apple Event Manager Result Codes" (page 252).

Discussion

Passing an argument list to `vAEBuildDesc` corresponds to passing a series of individual parameters to the `AEBuildDesc` (page 26) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple Event Manager routines to build up the descriptor piece by piece. The `vAEBuildDesc` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEHelpers.h`

vAEBuildParameters

Allows you to encapsulate calls to `AEBuildParameters` in your own `stdarg`-style wrapper routines, using techniques similar to those allowed by vsprintf.

```
OSStatus vAEBuildParameters (
   AppleEvent *event,
   AEBuildError *error,
   const char *format,
   va_list args
);
```

Parameters

event

The Apple event to which you are adding parameters. See `AppleEvent` (page 175).

error

A pointer to an `AEBuildError` structure where additional information about any errors that occur will be saved. This is an optional parameter and you can pass `NULL` if this information is not required. See `AEBuildError` (page 162).

format

An *AEBuild* format string describing the `AEDesc` parameters to be created.

args

A reference to a previously defined, variable argument parameter list to use with the descriptor-string. The file `<stdarg.h>` defines macros for declaring and using the `va_list` data type.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252).

Discussion
Passing an argument list to `vAEBuildParameters` corresponds to passing a series of individual parameters to the `AEBuildParameters` (page 27) function.

This function and related "AEBuild" routines provide a very simple translation service for converting specially formatted strings into complex Apple event descriptors. Normally, creating complex Apple event descriptors requires a large number of calls to Apple event Manager routines to build up the descriptor piece by piece. The `vAEBuildParameters` function and related routines allow you to consolidate all of the calls required to construct a complex Apple event descriptor into a single system call that creates the desired structure as directed by a format string that you provide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

Callbacks by Task

Callbacks When Resolving Remote Processes

`AERemoteProcessResolverCallback` (page 148)

Defines a pointer to a function the Apple Event Manager calls when the asynchronous execution of a remote process resolver completes, either due to success or failure, after a call to the `AERemoteProcessResolverScheduleWithRunLoop` function. Your callback function can use the reference passed to it to get the remote process information.

Callbacks When Creating Apple Events

`AEDisposeExternalProcPtr` (page 143)
Defines a pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function. Your callback function disposes of the buffer you originally passed to that function.

Callbacks When Sending Apple Events

`AEFilterProcPtr` (page 146)
Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your filter function determines which high-level events your application is willing to handle.

`AEIdleProcPtr` (page 147)
Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your idle function must handle update, null, operating-system, and activate events.

Coercing Apple Event Data Callbacks

`AECoerceDescProcPtr` (page 140)
Defines a pointer to a function that coerces data stored in a descriptor. Your descriptor coercion callback function coerces the data from the passed descriptor to the specified type, returning the coerced data in a second descriptor.

`AECoercePtrProcPtr` (page 141)
Defines a pointer to a function that coerces data stored in a buffer. Your pointer coercion callback routine coerces the data from the passed buffer to the specified type, returning the coerced data in a descriptor.

Handling Apple Events Callbacks

`AEEventHandlerProcPtr` (page 144)
Defines a pointer to a function that handles one or more Apple events. Your Apple event handler function performs any action requested by the Apple event, adds parameters to the reply Apple event if appropriate (possibly including error information), and returns a result code.

Object Accessor Callbacks

`OSLAccessorProcPtr` (page 149)
Your object accessor function either finds elements or properties of an Apple event object.

Object Callback Functions

`OSLAdjustMarksProcPtr` (page 151)
Defines a pointer to an adjust marks callback function. Your adjust marks function unmarks objects previously marked by a call to your marking function.

`OSLCompareProcPtr` (page 152)
Defines a pointer to an object comparison callback function. Your object comparison function compares one Apple event object to another or to the data for a descriptor.

`OSLCountProcPtr` (page 154)
Defines a pointer to an object counting callback function. Your object counting function counts the number of Apple event objects of a specified class in a specified container object.

`OSLDisposeTokenProcPtr` (page 155)
Defines a pointer to a dispose token callback function. Your dispose token function, required only if you use a complex token format, disposes of the specified token.

`OSLGetErrDescProcPtr` (page 157)
Defines a pointer to an error descriptor callback function. Your error descriptor callback function supplies a pointer to an address where the Apple Event Manager can store the current descriptor if an error occurs during a call to the `AEResolve` function.

`OSLGetMarkTokenProcPtr` (page 158)
Defines a pointer to a mark token callback function. Your mark token function returns a mark token.

`OSLMarkProcPtr` (page 160)
Defines a pointer to an object marking callback function. Your object-marking function marks a specific Apple event object.

Callbacks

AECoerceDescProcPtr

Defines a pointer to a function that coerces data stored in a descriptor. Your descriptor coercion callback function coerces the data from the passed descriptor to the specified type, returning the coerced data in a second descriptor.

```
typedef OSErr (*AECoerceDescProcPtr)
(
    const AEDesc * fromDesc,
    DescType toType,
    long handlerRefcon,
    AEDesc * toDesc
);
```

If you name your function `MyAECoerceDescCallback`, you would declare it like this:

```
OSErr MyAECoerceDescCallback (
    const AEDesc * fromDesc,
    DescType toType,
    long handlerRefcon,
    AEDesc * toDesc
);
```

Parameters

fromDesc

A pointer to the descriptor that contains the data to coerce. See AEDesc (page 162).

toType

The desired descriptor type for the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See DescType (page 176).

handlerRefcon

A reference constant that is stored in the coercion dispatch table entry for the handler. The Apple Event Manager passes this value to the handler each time it calls it. The reference constant may have a value of 0.

toDesc

A pointer to a descriptor where your coercion routine must store the descriptor that contains the coerced data. See AEDesc (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your handler should return noErr if it successfully handled the coercion, errAECoercionFailed if it can't handle the coercion and it wants the Apple Event Manager to continue dispatching to other coercion handlers, or a nonzero result code otherwise.

Discussion

Your coercion handler should coerce the data to the desired descriptor type and return the resulting data in the descriptor specified by the result parameter.

To provide a pointer to your descriptor coercion callback function, you create a universal procedure pointer (UPP) of type AECoerceDescUPP (page 168), using the function NewAECoerceDescUPP (page 130). You can do so with code like the following:

```
AECoerceDescUPP MyCoerceDescUPP;
MyCoerceDescUPP = NewAECoerceDescUPP (&MyCoerceDescCallback)
```

You can then pass the UPP MyCoerceDescUPP as a parameter to any function that installs or removes a coercion handler, such as AEInstallCoercionHandler (page 64). If your application installs the same coercion handler to coerce more than one type of data, you can use the same UPP to install the handler multiple times.

If you wish to call your descriptor coercion callback function directly, you can use the InvokeAECoerceDescUPP (page 123) function.

After you are finished with a descriptor coercion callback function, and have removed it with the AERemoveCoercionHandler (page 84) function, you can dispose of the UPP with the DisposeAECoerceDescUPP (page 119) function. However, don't dispose of the UPP if any remaining coercion handler uses it or if you plan to install the coercion handler again.

Availability

Available in Mac OS X v10.0 and later.

Declared In

AEDataModel.h

AECoercePtrProcPtr

Defines a pointer to a function that coerces data stored in a buffer. Your pointer coercion callback routine coerces the data from the passed buffer to the specified type, returning the coerced data in a descriptor.

```
typedef OSErr (*AECoercePtrProcPtr) (
    DescType typeCode,
    const void * dataPtr,
    Size dataSize,
    DescType toType,
    long handlerRefcon,
    AEDesc * result
);
```

If you name your function `MyAECoercePtrCallback`, you would declare it like this:

```
OSErr MyAECoercePtrCallback (
    DescType typeCode,
    const void * dataPtr,
    Size dataSize,
    DescType toType,
    long handlerRefcon,
    AEDesc * result
);
```

Parameters

typeCode

The descriptor type of the original data. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

dataPtr

A pointer to the data to coerce.

dataSize

The length, in bytes, of the data to coerce.

toType

The desired descriptor type for the resulting descriptor. For a list of AppleScript's predefined descriptor types, see "Descriptor Type Constants" (page 197). See `DescType` (page 176).

handlerRefcon

A reference constant that is stored in the coercion dispatch table entry for the handler. The Apple Event Manager passes this value to the handler each time it calls it. The reference constant may have a value of `NULL`.

result

A pointer to a descriptor where your coercion routine must store the descriptor that contains the coerced data. If your routine cannot coerce the data, return a null descriptor. See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your handler should return `noErr` if it successfully handled the coercion, `errAECoercionFailed` if it can't handle the coercion and it wants the Apple Event Manager to continue dispatching to other coercion handlers, or a nonzero result code otherwise.

Discussion

To provide a pointer to your coercion callback function, you create a universal procedure pointer (UPP) of type `AECoercePtrUPP` (page 168), using the function `NewAECoercePtrUPP` (page 130). You can do so with code like the following:

```
AECoercePtrUPP MyCoercePtrUPP;
MyCoercePtrUPP = NewAECoercePtrUPP (&MyCoercePtrCallback)
```

You can then pass the UPP `MyCoercePtrUPP` as a parameter to any function that installs or removes a coercion handler, such as `AEInstallCoercionHandler` (page 64). If your application installs the same coercion handler to coerce more than one type of data, you can use the same UPP to install the handler multiple times.

If you wish to call your coercion callback function directly, you can use the `InvokeAECoercePtrUPP` (page 124) function.

After you are finished with a coercion callback function, and have removed it with the `AERemoveCoercionHandler` (page 84) function, you can dispose of the UPP with the `DisposeAECoercePtrUPP` (page 119) function. However, don't dispose of the UPP if any remaining coercion handler uses it or if you plan to install the coercion handler again.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDisposeExternalProcPtr

Defines a pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function. Your callback function disposes of the buffer you originally passed to that function.

```
typedef (void, AEDisposeExternalProcPtr)(
                const void *dataPtr,
                Size dataLength,
                long refcon);
```

If you name your function `MyAEDisposeExternalCallback`, you would declare it like this:

```
void MyAEDisposeExternalCallback (
            const void *dataPtr,
            Size dataLength,
            long refcon);
```

Parameters

dataPtr

A pointer to the data to be disposed of. The data must be immutable and must not be freed until this callback function is called.

dataLength

The length, in bytes, of the data in the *dataPtr* parameter.

refcon

A reference constant, supplied by your application in its original call to `AECreateDescFromExternalPtr` (page 34). The Apple Event Manager passes this value to your dispose function each time it calls it. The reference constant may have a value of 0.

Return Value
Your callback routine should not return a value.

Discussion
Your application must provide a universal procedure pointer to a dispose function as a parameter to the `AECreateDescFromExternalPtr` (page 34) function.

To provide a pointer to your dispose callback function, you create a universal procedure pointer (UPP) of type `AEDisposeExternalProcPtr`, using the function `NewAEDisposeExternalUPP` (page 130). You can do so with code like the following:

```
AEDisposeExternalProcPtr MyDisposeCallbackUPP;
MyDisposeCallbackUPP = NewAEDisposeExternalUPP (&MyAEDisposeExternalCallback);
```

You can then pass the UPP `MyDisposeCallbackUPP` as a parameter to the `AECreateDescFromExternalPtr` function.

If you wish to call your dispose callback function directly, you can use the `InvokeAEDisposeExternalUPP` (page 124) function.

After you are finished with your dispose callback function, you can dispose of the UPP with the `DisposeAEDisposeExternalUPP` (page 119) function. However, if you will use the same dispose function in subsequent calls to `AECreateDescFromExternalPtr`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AEEventHandlerProcPtr

Defines a pointer to a function that handles one or more Apple events. Your Apple event handler function performs any action requested by the Apple event, adds parameters to the reply Apple event if appropriate (possibly including error information), and returns a result code.

```
typedef OSErr (*AEEventHandlerProcPtr)
(
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    long handlerRefcon
);
```

If you name your function `MyAEEventHandlerCallback`, you would declare it like this:

```
OSErr MyAEEventHandlerCallback (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    long handlerRefcon
);
```

Parameters

`theAppleEvent`
A pointer to the Apple event to handle. See `AppleEvent` (page 175).

reply

A pointer to the default reply Apple event provided by the Apple Event Manager. See `AppleEvent` (page 175). If no reply is expected, `reply` has descriptor type `typeNull`.

handlerRefcon

The reference constant stored in the Apple event dispatch table when you install the handler function for the Apple event. You can store any 32-bit value in the dispatch table and use it any way you want when the handler is called. The reference constant may have a value of `NULL`.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your handler should always return `noErr` if it successfully handled the Apple event. If an error occurs, your handler should return either `errAEEventNotHandled` or some other nonzero result code. For more information, see the Discussion section.

Discussion

An Apple event handler should extract any parameters and attributes from the Apple event, perform the requested action, and add parameters to the reply Apple event if appropriate. You must provide an Apple event handler for each Apple event your application supports. The `AEProcessAppleEvent` (page 73) function calls one of your Apple event handlers when it processes an Apple event.

If an error occurs because your application cannot understand the event, return `errAEEventNotHandled`, so that the Apple Event Manager may be able to find another handler to handle the event. If the error occurs because the event is impossible to handle as specified, return the result code returned by whatever function caused the failure, or whatever other result code is appropriate.

For example, suppose your application receives a `kAEGetData` event that requests the name of the current printer, and your application cannot handle such an event. In this situation, you should return `errAEEventNotHandled` so that another handler available to the Apple Event Manager can have a chance to handle the event. This strategy allows users to take advantage of system capabilities from within your application via system handlers.

If your Apple event handler calls the `AEResolve` (page 89) function and `AEResolve` calls an object accessor function in the system object accessor dispatch table, your Apple event handler may not recognize the descriptor type of the token returned by the function. In this case, your handler should return the result code `errAEUnknownObjectType`. When your handler returns this result code, the Apple Event Manager attempts to locate a system Apple event handler that can recognize the token.

For additional information on dealing with error conditions, see `OSLGetErrDescProcPtr` (page 157).

To provide a pointer to your event handler callback function, you create a universal procedure pointer (UPP) of type `AEEventHandlerUPP` (page 171), using the function `NewAEEventHandlerUPP` (page 131). You can do so with code like the following:

```
AEEventHandlerUPP MyEventHandlerUPP;
MyEventHandlerUPP = NewAEEventHandlerUPP (&MyEventHandlerCallback)
```

You can then pass the UPP `MyEventHandlerUPP` as a parameter to any function that installs or removes a handler, such as `AEInstallEventHandler` (page 65). If your application installs the same event handler to handle more than one kind of event (more than one pair of event class and event ID), you can use the same UPP to install the handler multiple times.

If you wish to call your event handler callback function directly, you can use the `InvokeAEEventHandlerUPP` (page 125) function.

After you are finished with an event handler callback function, and have removed it with the `AERemoveEventHandler` (page 85) function, you can dispose of the UPP with the `DisposeAEEventHandlerUPP` (page 120) function. However, don't dispose of the UPP if any remaining handler uses it or if you plan to install the handler again.

Version Notes
Your application should not install a handler in a system dispatch table with the goal that the handler will get called when other applications receive an Apple event—this won't work in Mac OS X. For more information, see "The System Dispatch Table" in "Apple Event Dispatching" in Apple Events Programming Guide.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFilterProcPtr

Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your filter function determines which high-level events your application is willing to handle.

```
typedef Boolean (*AEFilterProcPtr) (
    EventRecord * theEvent,
    long returnID,
    long transactionID,
    const AEAddressDesc * sender
);
```

If you name your function `MyAEFilterCallback`, you would declare it like this:

```
Boolean MyAEFilterCallback (
    EventRecord * theEvent,
    long returnID,
    long transactionID,
    const AEAddressDesc * sender
);
```

Parameters

theEvent
A pointer to the event record for a high-level event. The next three parameters contain valid information only if the event is an Apple event. See the Event Manager documentation for a description of the `EventRecord` data type.

returnID
Return ID for the Apple event.

transactionID
Transaction ID for the Apple event.

sender
A pointer to the address of the process that sent the Apple event. See `AEAddressDesc` (page 167).

Return Value
Your filter routine returns `TRUE` to accept the Apple event or `FALSE` to filter it out.

Discussion
If your application provides a universal procedure pointer to a reply filter function as a parameter to the `AESend` (page 92) function, the reply filter function can indicate any high-level events that it is willing to handle while your application is waiting for a reply.

If your filter function returns `true`, the Apple Event Manager will dispatch the event through the standard dispatch mechanism (equivalent to calling `AEProcessAppleEvent` (page 73)).

To provide a pointer to your reply filter callback function, you create a universal procedure pointer (UPP) of type `AEFilterUPP` (page 172), using the function `NewAEFilterUPP` (page 131). You can do so with code like the following:

```
AEFilterUPP MyReplyFilterUPP;
MyReplyFilterUPP = NewAEFilterUPP (&MyReplyFilterCallback)
```

You can then pass the UPP `MyReplyFilterUPP` as a parameter to the `AESend` function.

If you wish to call your filter callback function directly, you can use the `InvokeAEFilterUPP` (page 125) function.

After you are finished with your filter callback function, you can dispose of the UPP with the `DisposeAEFilterUPP` (page 120) function. However, if you will use the same filter function in subsequent calls to `AESend`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEIdleProcPtr

Defines a pointer to a function the Apple Event Manager calls while your application waits for a reply to an Apple event. Your idle function must handle update, null, operating-system, and activate events.

```
typedef Boolean (*AEIdleProcPtr) (
    EventRecord * theEvent,
    long * sleepTime,
    RgnHandle * mouseRgn
);
```

If you name your function `MyAEIdleCallback`, you would declare it like this:

```
Boolean MyAEIdleCallback (
    EventRecord * theEvent,
    long * sleepTime,
    RgnHandle * mouseRgn
);
```

Parameters

`theEvent`

A pointer to the event record of the event to process. See the Event Manager documentation for a description of the `EventRecord` data type.

sleepTime

A pointer to a value that specifies the amount of time (in ticks) your application is willing to relinquish the processor if no events are pending.

mouseRgn

A pointer to a value that specifies a screen region that determines the conditions under which your application is to receive notice of mouse-moved events. See the QuickDraw Manager documentation for a description of the `RgnHandle` data type.

Return Value

Your idle routine returns `TRUE` if your application is no longer willing to wait for a reply from the server or for the user to bring the application to the front. It returns `FALSE` if your application is still willing to wait.

Discussion

If your application provides a pointer to an idle function as a parameter to the `AESend` (page 92) function or the `AEInteractWithUser` (page 69) function, the Apple Event Manager will call the idle function to handle any update event, null event, operating-system event, or activate event received for your application while it is waiting for a reply.

To provide a pointer to your idle callback function, you create a universal procedure pointer (UPP) of type `AEIdleUPP` (page 172), using the function `NewAEIdleUPP` (page 131). You can do so with code like the following:

```
AEIdleUPP MyIdleUPP;
MyIdleUPP = NewAEIdleUPP (&MyIdleCallback)
```

You can then pass the UPP `MyIdleUPP` as a parameter to either the `AESend` function or the `AEInteractWithUser` function.

If you wish to call your idle callback function directly, you can use the `InvokeAEIdleUPP` (page 125) function.

After you are finished with your idle callback function, you can dispose of the UPP with the `DisposeAEIdleUPP` (page 120) function. However, if you will use the same idle function in subsequent calls to `AESend` or `AEInteractWithUser`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AEInteraction.h`

AERemoteProcessResolverCallback

Defines a pointer to a function the Apple Event Manager calls when the asynchronous execution of a remote process resolver completes, either due to success or failure, after a call to the `AERemoteProcessResolverScheduleWithRunLoop` function. Your callback function can use the reference passed to it to get the remote process information.

```
typedef (void, AERemoteProcessResolverCallback)(
                AERemoteProcessResolverRef ref,
                void *info);
```

If you name your function `MyAERemoteProcessCallback`, you would declare it like this:

```
void MyAERemoteProcessCallback (
```

```
            AERemoteProcessResolverRef ref,
            void *info);
```

Parameters

ref

A reference of type `AERemoteProcessResolverRef` (page 173) you can query to obtain the remote process information. Acquired from a previous call to `AECreateRemoteProcessResolver` (page 36).

info

An untyped pointer your application can use to pass information it needs when resolving remote processes. The application originally supplies this pointer in the `AERemoteProcessResolverContext` (page 163) structure in the `ctx` parameter) when it calls the `AERemoteProcessResolverScheduleWithRunLoop` function.

Return Value
Your callback routine should not return a value.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

OSLAccessorProcPtr

Your object accessor function either finds elements or properties of an Apple event object.

```
typedef OSErr (*OSLAccessorProcPtr) (
    DescType desiredClass,
    const AEDesc * container,
    DescType containerClass,
    DescType form,
    const AEDesc * selectionData,
    AEDesc * value,
    long accessorRefcon
);
```

If you name your function `MyObjectAccessorCallback`, you would declare it like this:

```
OSErr MyObjectAccessorCallback (
    DescType desiredClass,
    const AEDesc * container,
    DescType containerClass,
    DescType form,
    const AEDesc * selectionData,
    AEDesc * value,
    long accessorRefcon
);
```

Parameters

desiredClass

The object class of the desired Apple event object or objects. Constants for object class IDs are described in "Object Class ID Constants" (page 215). See `DescType` (page 176).

container

A pointer to a descriptor that specifies the container of the desired Apple event object or objects. See AEDesc (page 162).

containerClass

The object class of the container. Constants for object class IDs are described in "Object Class ID Constants" (page 215). See DescType (page 176).

form

The key form specified by the object specifier being resolved. Constants for key form are described in "Key Form and Descriptor Type Object Specifier Constants" (page 206). See DescType (page 176).

selectionData

A pointer to a descriptor containing the key data specified by the object specifier being resolved. See AEDesc (page 162).

value

A pointer to a descriptor where your object accessor routine stores a descriptor that identifies the found object. See AEDesc (page 162).

accessorRefcon

A reference constant. The Apple Event Manager passes this value to your object accessor function each time it calls it. The reference constant may have a value of 0.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your object accessor function should return noErr if it successfully located the requested object and errAEEventNotHandled if it could not locate the object. When the Apple Event Manager receives the result code errAEEventNotHandled after calling an object accessor function, it attempts to use other methods of locating the requested objects, such as calling an equivalent system object accessor function.

Discussion

To resolve an object specifier, your application calls the AEResolve (page 89) function. AEResolve in turn calls application-defined object accessor functions to locate specific Apple event objects and properties in the application's data structures. Your application provides one or more object accessor functions that can locate all the element classes and properties it supports.

Each object accessor function provided by your application should either find elements or properties of an Apple event object. The AEResolve function uses the object class ID of the specified Apple event object and the descriptor type of the token that identifies the object's container to determine which object accessor function to call. To install an object accessor function, use the AEInstallObjectAccessor (page 67) function.

To provide a pointer to your object accessor callback function, you create a universal procedure pointer (UPP) of type OSLAccessorUPP (page 176), using the function NewOSLAccessorUPP (page 132). You can do so with code like the following:

```
AEObjectAccessorUPP MyObjectAccessorUPP;
MyObjectAccessorUPP = NewAEObjectAccessorUPP (&MyObjectAccessorCallback)
```

You can then pass the UPP MyObjectAccessorUPP as a parameter to any function that installs or removes an object accessor, such as AEInstallObjectAccessor (page 67). If your application installs the same object accessor to handle more than one kind of object class or property of an Apple event, you can use the same UPP to install the accessor multiple times.

If you wish to call your object accessor callback function directly, you can use the InvokeOSLAccessorUPP (page 126) function.

After you are finished with an object accessor callback function, and have removed it with the `AERemoveObjectAccessor` (page 86) function, you can dispose of the UPP with the `DisposeOSLAccessorUPP` (page 120) function. However, don't dispose of the UPP if any remaining accessor function uses it or if you plan to install the accessor function again.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLAdjustMarksProcPtr

Defines a pointer to an adjust marks callback function. Your adjust marks function unmarks objects previously marked by a call to your marking function.

```
typedef OSErr (*OSLAdjustMarksProcPtr)
(
    long newStart,
    long newStop,
    const AEDesc * markToken
);
```

If you name your function `MyAdjustMarksCallback`, you would declare it like this:

```
OSErr MyAdjustMarksCallback (
    long newStart,
    long newStop,
    const AEDesc * markToken
);
```

Parameters

`newStart`
The mark count value (provided when the `MyAdjustMarksCallback` callback function was called to mark the object) for the first object in the new set of marked objects.

`newStop`
The mark count value (provided when the `MyAdjustMarksCallback` callback function was called to mark the object) for the last object in the new set of marked objects.

`markToken`
A pointer to the mark token for the marked objects. (Token is defined in `AEDisposeToken` (page 41). See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). Your adjust marks function should return `noErr` if it successfully adjusted the marks and `errAEEventNotHandled` if it could not locate the object. When the Apple Event Manager gets an error result of `errAEEventNotHandled`, it attempts to adjust the marks by calling the equivalent system mark-adjusting function.

Discussion
When the Apple Event Manager needs to identify either a range of elements or the absolute position of an element in a group of Apple event objects that pass a test, it can use your application's mark-adjusting function to unmark objects previously marked by a call to your marking function.

For example, suppose an object specifier specifies `any row in the table "MyCustomers" for which the City column is "San Francisco"`. The Apple Event Manager first uses the appropriate object accessor function to locate all the rows in the table for which the City column is `"San Francisco"` and calls the application's marking function repeatedly to mark them. It then generates a random number between 1 and the number of rows it found that passed the test and calls the application's mark-adjusting function to unmark all the rows whose mark count does not match the randomly generated number. If the randomly chosen row has a mark count value of 5, the Apple Event Manager passes the value 5 to the mark-adjusting function in both the `newStart` parameter and the `newStop` parameter, and passes the current mark token in the `markToken` parameter.

When the Apple Event Manager calls your `MyAdjustMarksCallback` function, your application must dispose of any data structures that it created to mark the previously marked objects.

To provide a pointer to your adjust marks callback function, you create a universal procedure pointer (UPP) of type `OSLAdjustMarksUPP` (page 177), using the function `NewOSLAdjustMarksUPP` (page 132). You can do so with code like the following:

```
OSLAdjustMarksUPP MyAdjustMarksUPP;
MyAdjustMarksUPP = NewOSLAdjustMarksUPP (&MyAdjustMarksCallback)
```

You can then pass the UPP `MyAdjustMarksUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your adjust marks callback function directly, you can use the `InvokeOSLAdjustMarksUPP` (page 126) function.

After you are finished with your adjust marks callback function, you can dispose of the UPP with the `DisposeOSLAdjustMarksUPP` (page 121) function. However, if you will use the same adjust marks function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCompareProcPtr

Defines a pointer to an object comparison callback function. Your object comparison function compares one Apple event object to another or to the data for a descriptor.

```
typedef OSErr (*OSLCompareProcPtr) (
    DescType oper,
    const AEDesc * obj1,
    const AEDesc * obj2,
    Boolean * result
);
```

If you name your function `MyCompareObjectsCallback`, you would declare it like this:

```
OSErr MyCompareObjectsCallback (
    DescType oper,
    const AEDesc * obj1,
    const AEDesc * obj2,
    Boolean * result
);
```

Parameters

oper

A comparison operator that specifies the type of comparison to perform. The available comparison operators are described in "Comparison Operator Constants" (page 190). For related information, see the function `CreateCompDescriptor` (page 114). See `DescType` (page 176).

obj1

A pointer to a token describing the first Apple event object to compare. (Token is defined in `AEDisposeToken` (page 41). See `AEDesc` (page 162).

obj2

A pointer to a token or some other descriptor that specifies either an Apple event object or a value to compare to the Apple event object specified by the `obj1` parameter. See `AEDesc` (page 162).

result

A pointer to a Boolean value where your object comparison function stores a value indicating the result of the comparison operation. You store `TRUE` if the values of the `obj1` and `obj2` parameters have the relationship specified by the `comparisonOperator` parameter; otherwise, you store `FALSE`.

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your object comparison function should return `noErr` if it successfully compared the objects and `errAEEventNotHandled` if it can't compare the objects. When the Apple Event Manager gets an error result of `errAEEventNotHandled`, it attempts to use other methods of comparing the specified objects, such as calling an equivalent system object comparison function.

Discussion

The Apple Event Manager calls your object comparison function when, in the course of resolving an object specifier, the manager needs to compare an Apple event object with another object or with a value in a descriptor.

If you want the Apple Event Manager to help your application resolve object specifiers of key form `formTest` (and if your application doesn't specify `kAEIDoWhose` as described in "Callback Constants for the AEResolve Function" (page 187)), you should provide an object-counting function, as described in `OSLCountProcPtr` (page 154), and an object comparison function.

It is up to your application to interpret the comparison operators it receives. The meaning of comparison operators differs according to the Apple event objects being compared, and not all comparison operators apply to all object classes. The available comparison operators are described in "Comparison Operator Constants" (page 190).

To provide a pointer to your object comparison callback function, you create a universal procedure pointer (UPP) of type `OSLCompareUPP` (page 177), using the function `NewOSLCompareUPP` (page 133). You can do so with code like the following:

```
OSLCompareObjectsUPP MyCompareObjectsUPP;
MyCompareObjectsUPP = NewOSLCompareObjectsUPP(&MyCompareObjectsCallback)
```

You can then pass the UPP `MyCompareObjectsUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your object comparison callback function directly, you can use the `InvokeOSLCompareUPP` (page 127) function.

After you are finished with your object comparison callback function, you can dispose of the UPP with the `DisposeOSLCompareUPP` (page 121) function. However, if you will use the same object comparison function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCountProcPtr

Defines a pointer to an object counting callback function. Your object counting function counts the number of Apple event objects of a specified class in a specified container object.

```
typedef OSErr (*OSLCountProcPtr) (
    DescType desiredType,
    DescType containerClass,
    const AEDesc * container,
    long * result
);
```

If you name your function `MyCountObjectsCallback`, you would declare it like this:

```
OSErr MyCountObjectsCallback (
    DescType desiredType,
    DescType containerClass,
    const AEDesc * container,
    long * result
);
```

Parameters

desiredType
The object class of the Apple event objects to be counted. See `DescType` (page 176).

containerClass
The object class of the container for the Apple event objects to be counted. See `DescType` (page 176).

container
A pointer to a token that identifies the container for the Apple event objects to be counted. (Token is defined in `AEDisposeToken` (page 41). See `AEDesc` (page 162).

result

A pointer to a variable where your object-counting function stores the number of Apple objects of the specified class in the specified container.

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). Your object counting function should return `noErr` if it successfully counted the objects and `errAEEventNotHandled` if it can't count the objects. When the Apple Event Manager receives the result code `errAEEventNotHandled` after calling an object counting function, it attempts to use other methods of counting the specified objects, such as calling an equivalent system object counting function.

Discussion
If you want the Apple Event Manager to help your application resolve object specifiers of key form `formTest` (and if your application doesn't specify `kAEIDoWhose` as described in "Callback Constants for the AEResolve Function" (page 187)), you should provide an object comparison function, as described in `OSLCompareProcPtr` (page 152), and an object-counting function.

The Apple Event Manager calls your object-counting function when, in the course of resolving an object specifier, the manager requires a count of the number of Apple event objects of a given class in a given container.

To provide a pointer to your object counting callback function, you create a universal procedure pointer (UPP) of type `OSLCountUPP` (page 177), using the function `NewOSLCountUPP` (page 133). You can do so with code like the following:

```
OSLCountObjectsUPP MyCountObjectsUPP;
MyCountObjectsUPP = NewOSLCountObjectsUPP (&MyCountObjectsCallback)
```

You can then pass the UPP `MyCountObjectsUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your object counting callback function directly, you can use the `InvokeOSLCountUPP` (page 127) function.

After you are finished with your object counting callback function, you can dispose of the UPP with the `DisposeOSLCountUPP` (page 121) function. However, if you will use the same object counting function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLDisposeTokenProcPtr

Defines a pointer to a dispose token callback function. Your dispose token function, required only if you use a complex token format, disposes of the specified token.

```
typedef OSErr (*OSLDisposeTokenProcPtr)
(
    AEDesc * unneededToken
);
```

If you name your function `MyDisposeTokenCallback`, you would declare it like this:

```
OSErr MyDisposeTokenCallback (
    AEDesc * unneededToken
);
```

Parameters

unneededToken

A pointer to the token to dispose of. (Token is defined in `AEDisposeToken` (page 41).) On successful return, your function must set this to the null descriptor. See `AEDesc` (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your token disposal function should return `noErr` if it successfully disposed of the token and `errAEEventNotHandled` if it can't dispose of the token. When the Apple Event Manager receives the result code `errAEEventNotHandled` after calling a token disposal function, it attempts to use other methods of disposing of the specified token, such as calling an equivalent system token disposal function if one is available or, if that fails, by calling `AEDisposeDesc` (page 40).

Discussion

The Apple Event Manager calls your token disposal function whenever it needs to dispose of a token. It also calls your disposal function when your application calls the `AEDisposeToken` (page 41) function. If your application does not provide a token disposal function, the Apple Event Manager calls `AEDisposeDesc` (page 40) instead.

Your token disposal function must be able to dispose of all of the token types used by your application.

If your application supports marking, a call to `MyDisposeTokenCallback` to dispose of a mark token lets your application know that it can unmark the objects marked with that mark token, as described in the Discussion section for `OSLGetMarkTokenProcPtr` (page 158).

To provide a pointer to your token disposal callback function, you create a universal procedure pointer (UPP) of type `OSLDisposeTokenUPP` (page 177), using the function `NewOSLDisposeTokenUPP` (page 133). You can do so with code like the following:

```
OSLDisposeTokenUPP MyDisposeTokenUPP;
MyDisposeTokenUPP = NewOSLDisposeTokenUPP (&MyDisposeTokenCallback)
```

You can then pass the UPP `MyDisposeTokenUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your token disposal callback function directly, you can use the `InvokeOSLDisposeTokenUPP` (page 128) function.

After you are finished with your token disposal callback function, you can dispose of the UPP with the `DisposeOSLDisposeTokenUPP` (page 122) function. However, if you will use the same token disposal function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Version Notes
In Mac OS X, your application can not make an object callback function available to other applications by installing it in a system object accessor dispatch table.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetErrDescProcPtr

Defines a pointer to an error descriptor callback function. Your error descriptor callback function supplies a pointer to an address where the Apple Event Manager can store the current descriptor if an error occurs during a call to the `AEResolve` function.

```
typedef OSErr (*OSLGetErrDescProcPtr)
(
    AEDesc ** appDescPtr
);
```

If you name your function `MyGetErrorDescCallback`, you would declare it like this:

```
OSErr MyGetErrorDescCallback (
    AEDesc ** appDescPtr
);
```

Parameters

`appDescPtr`

A pointer to a pointer to a descriptor address. Your error descriptor callback function supplies a pointer to an address of a descriptor where the Apple Event Manager can store the current descriptor if an error occurs. See `AEDesc` (page 162).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). Your error descriptor function should return `noErr` if it completes successfully and a nonzero error value if it is unsuccessful. If it returns a nonzero value, the Apple Event Manager continues to resolve the object specifier as if it had never called the error callback function.

Discussion
Your get error descriptor callback function simply supplies a pointer to an address. Shortly after your application calls the `AEResolve` (page 89) function, the Apple Event Manager calls your get error descriptor callback function and writes a null descriptor to the address supplied by your callback, overwriting whatever was there previously.

If an error occurs during the resolution of the object specifier, the Apple Event Manager calls your get error descriptor callback function again and writes the descriptor it is currently working with—often an object specifier—to the address supplied by your callback. If `AEResolve` returns an error during the resolution of an object specifier, this address contains the descriptor responsible for the error.

You should always write a null descriptor at the address provided by your get error descriptor callback function before calling `AEResolve`. When recovering from an error, the Apple Event Manager, never writes to the address you provide unless it already contains a null descriptor. You may wish to maintain a single global variable of type `AEDesc` and have your get error descriptor callback function always provide the address of that variable.

After `AEResolve` returns, if your error descriptor is not the null descriptor, you are responsible for disposing of it.

To provide a pointer to your get error descriptor callback function, you create a universal procedure pointer (UPP) of type `OSLGetErrDescUPP` (page 178), using the function `NewOSLGetErrDescUPP` (page 134). You can do so with code like the following:

```
OSLGetErrorDescUPP MyGetErrorDescUPP;
MyGetErrorDescUPP = NewOSLGetErrorDescUPP (&MyGetErrorDescCallback)
```

You can then pass the UPP `MyGetErrorDescUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your get error descriptor callback function directly, you can use the `InvokeOSLGetErrDescUPP` (page 128) function.

After you are finished with your get error descriptor callback function, you can dispose of the UPP with the `DisposeOSLGetErrDescUPP` (page 122) function. However, if you will use the same get error descriptor callback function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetMarkTokenProcPtr

Defines a pointer to a mark token callback function. Your mark token function returns a mark token.

```
typedef OSErr (*OSLGetMarkTokenProcPtr)
(
    const AEDesc * dContainerToken,
    DescType containerClass,
    AEDesc * result
);
```

If you name your function `MyGetMarkTokenCallback`, you would declare it like this:

```
OSErr MyGetMarkTokenCallback (
    const AEDesc * dContainerToken,
    DescType containerClass,
    AEDesc * result
);
```

Parameters

dContainerToken

A pointer to the Apple event object that contains the elements to be marked with the mark token. (Token is defined in AEDisposeToken (page 41). See AEDesc (page 162).

containerClass

The object class of the container that contains the objects to be marked. See DescType (page 176).

result

A pointer to a descriptor where your mark token function should return a mark token. If your function can't return a mark token, it should return a null descriptor. See AEDesc (page 162).

Return Value

A result code. See "Apple Event Manager Result Codes" (page 252). Your mark token function should return noErr if it successfully supplies a mark token and errAEEventNotHandled if it fails to supply a mark token. When the Apple Event Manager gets an error result of errAEEventNotHandled after calling a mark token function, it attempts to get a mark token by calling the equivalent system marking callback function.

Discussion

To get a mark token, the Apple Event Manager calls your mark token function. Like other tokens, the mark token returned can be a descriptor of any type; however, unlike other tokens, a mark token identifies the way your application will mark Apple event objects during the current session while resolving a single object specifier that specifies the key form formTest.

A mark token is valid until the Apple Event Manager either disposes of it by calling AEDisposeToken (page 41) or returns it as the result of the AEResolve (page 89) function. If the final result of a call to AEResolve is a mark token, the Apple event objects currently marked for that mark token are those specified by the object specifier passed to AEResolve, and your application can proceed to do whatever the Apple event has requested. Note that your application is responsible for disposing of a final mark token with a call to AEDisposeToken, just as for any other final token.

If your application supports marking, it should also provide a token disposal function modeled after the token disposal function described in OSLDisposeTokenProcPtr (page 155). When the Apple Event Manager calls AEDisposeToken to dispose of a mark token that is not the final result of a call to AEResolve, the subsequent call to your token disposal function lets you know that you can unmark the Apple event objects marked with that mark token. A call to AEDisposeDesc to dispose of a mark token (which would occur if you did not provide a token disposal function) would go unnoticed.

To provide a pointer to your mark token callback function, you create a universal procedure pointer (UPP) of type OSLGetMarkTokenUPP (page 178), using the function NewOSLGetMarkTokenUPP (page 134). You can do so with code like the following:

```
OSLGetMarkTokenUPP MyGetMarkTokenUPP;
MyGetMarkTokenUPP = NewOSLGetMarkTokenUPP (&MyGetMarkTokenCallback)
```

You can then pass the UPP MyGetMarkTokenUPP as a parameter to the AESetObjectCallbacks (page 96) function or the AEInstallSpecialHandler (page 68) function.

If you wish to call your mark token callback function directly, you can use the InvokeOSLGetMarkTokenUPP (page 129) function.

After you are finished with your mark token callback function, you can dispose of the UPP with the DisposeOSLGetMarkTokenUPP (page 122) function. However, if you will use the same mark token function in subsequent calls to the function AESetObjectCallbacks or the function AEInstallSpecialHandler, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEObjects.h

OSLMarkProcPtr

Defines a pointer to an object marking callback function. Your object-marking function marks a specific Apple event object.

```
typedef OSErr (*OSLMarkProcPtr) (
    const AEDesc * dToken,
    const AEDesc * markToken,
    long index
);
```

If you name your function MyMarkCallback, you would declare it like this:

```
OSErr MyMarkCallback (
    const AEDesc * dToken,
    const AEDesc * markToken,
    long index
);
```

Parameters

dToken

A pointer to the token for the Apple event object to be marked. (Token is defined in AEDisposeToken (page 41). See AEDesc (page 162).

markToken

A pointer to the mark token used to mark the Apple event object. See AEDesc (page 162).

index

The number of times your MyMarkCallback function has been called for the current mark token (that is, the number of Apple event objects that have so far passed the test, including the element to be marked).

Return Value
A result code. See "Apple Event Manager Result Codes" (page 252). Your object marking function should return noErr if it successfully marks the Apple event object and errAEEventNotHandled if it fails to mark the object. When the Apple Event Manager gets an error result of errAEEventNotHandled after calling an object marking function, it attempts to get mark the object by calling the equivalent system object marking function.

Discussion
To mark an Apple event object using the current mark token, the Apple Event Manager calls the object-marking function provided by your application. In addition to marking the specified object, your MyMarkCallback function should record the mark count for each object that it marks. The mark count recorded for each marked object allows your application to determine which of a set of marked tokens pass a test, as described in the Discussion section for the OSLAdjustMarksProcPtr (page 151) function.

To provide a pointer to your mark callback function, you create a universal procedure pointer (UPP) of type OSLMarkUPP (page 178), using the function NewOSLMarkUPP (page 134). You can do so with code like the following:

```
OSLMarkUPP MyMarkUPP;
MyMarkUPP = NewOSLMarkUPP (&MyMarkCallback)
```

You can then pass the UPP `MyMarkUPP` as a parameter to the `AESetObjectCallbacks` (page 96) function or the `AEInstallSpecialHandler` (page 68) function.

If you wish to call your mark callback function directly, you can use the `InvokeOSLMarkUPP` (page 129) function.

After you are finished with your mark callback function, you can dispose of the UPP with the `DisposeOSLMarkUPP` (page 123) function. However, if you will use the same mark function in subsequent calls to the function `AESetObjectCallbacks` or the function `AEInstallSpecialHandler`, you can reuse the same UPP, rather than dispose of it and later create a new UPP.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

Data Types

AEArrayData

Stores array information to be put into a descriptor list with the `AEPutArray` function or extracted from a descriptor list with the `AEGetArray` function.

```
union AEArrayData {
    short kAEDataArray[1];
    char kAEPackedArray[1];
    Handle kAEHandleArray[1];
    AEDesc kAEDescArray[1];
    AEKeyDesc kAEKeyDescArray[1];
};
typedef union AEArrayData AEArrayData;
```

Discussion
When your application calls the `AEPutArray` (page 75) function to put information into a descriptor list or the `AEGetArray` (page 44) function to get information from a descriptor list, it uses an to store the information. The type of array depends on the data for the array, as specified by one of the constants described in "Data Array Constants" (page 196).

Array items in Apple event arrays of type `kAEDataArray`, `kAEPackedArray`, or `kAEHandleArray` must be factored—that is, contained in a factored descriptor list. Before adding array items to a factored descriptor list, you should provide both a pointer to the data that is common to all array items and the size of that common data when you first call `AECreateList` (page 35) to create a factored descriptor list. When you call `AEPutArray` to add the array data to such a descriptor list, the Apple Event Manager automatically isolates the common data you specified in the call to `AECreateList`.

When you call `AEGetArray` or `AEPutArray`, you specify a pointer of data type `AEArrayDataPointer` that points to a buffer containing the data for the array.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEBuildError

Defines a structure for storing additional error code information for "AEBuild" routines.

```
struct AEBuildError {
    AEBuildErrorCode fError;
    UInt32 fErrorPos;
};
typedef struct AEBuildError AEBuildError;
```

Fields

`fError`
The error code. See "AEBuild Error Codes" (page 179) for a list of errors.

`fErrorPos`
The character position where the parser detected the error.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEHelpers.h

AEDesc

Stores data and an accompanying descriptor type to form the basic building block of all Apple Events.

```
struct AEDesc {
    DescType descriptorType;
    AEDataStorage dataHandle;
};
typedef struct AEDesc AEDesc;
```

Fields

`descriptorType`
A four-character code of type `DescType` (page 176) that indicates the type of data in the structure. See `DescType` (page 176).

`dataHandle`
An opaque storage type that points to the storage for the descriptor data. Your application doesn't access this data directly—rather, it calls one of the functions `AEGetDescDataSize` (page 50), `AEGetDescData` (page 48), or `AEReplaceDescData` (page 88). See `AEDataStorage` (page 169).

Discussion
The Apple Event Manager uses one or more descriptors to construct Apple event attributes and parameters, object specifiers, tokens, and many other types of data it works with. (Token is defined in `AEDisposeToken` (page 41).) A descriptor consists of an opaque data storage container and a descriptor type that identifies the type of the data stored in the descriptor.

The descriptor type is a structure of type `DescType`, which in turn is of data type `ResType`—that is, a four-character code. "Descriptor Type Constants" (page 197) lists the constants for the basic descriptor types used by the Apple Event Manager. For information about descriptor types used with object specifiers, see "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Version Notes
Prior to Carbon, the `AEDataStorage` (page 169) data type was defined as follows:

```
typedef Handle AEDataStorage;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEKeyDesc

Associates a keyword with a descriptor to form a keyword-specified descriptor.

```
struct AEKeyDesc {
    AEKeyword descKey;
    AEDesc descContent;
};
typedef struct AEKeyDesc AEKeyDesc;
```

Fields

`descKey`
: A four-character code of type `AEKeyword` (page 172) that uniquely identifies the key that is associated with the data in the structure. Some keyword constants are described in "Keyword Attribute Constants" (page 209) and "Keyword Parameter Constants" (page 211). See `AEKeyword` (page 172).

`descContent`
: A descriptor of type `AEDesc` (page 162) that stores the keyword descriptor data. See `AEDesc` (page 162).

Discussion
The Apple Event Manager uniquely identifies the various parts of an Apple event by means of keywords associated with corresponding descriptors. A keyword is an arbitrary constant of type `AEKeyword` (page 172) that represents a four-character code.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoteProcessResolverContext

Supplied as a parameter when performing asynchronous resolution of remote processes.

```
struct AERemoteProcessResolverContext {
    CFIndex version;
    void * info;
    CFAllocatorRetainCallBack retain;
    CFAllocatorReleaseCallBack release;
    CFAllocatorCopyDescriptionCallBack copyDescription;
};
typedef struct AERemoteProcessResolverContext AERemoteProcessResolverContext;
```

Fields

`version`

This should be set to zero (0).

`info`

A pointer to arbitrary information. The pointer is retained and passed to the callback, allowing you to provide information to that routine.

`retain`

A prototype for a function callback that retains the specified data. Called on the info pointer. This field may be `NULL`.

`release`

A prototype for a function callback that releases the specified data. Called on the info pointer. This field may be `NULL`.

`copyDescription`

A prototype for a function callback that provides a description of the specified data. Called on the info pointer. This field may be `NULL`.

Discussion

When you call `AERemoteProcessResolverScheduleWithRunLoop` (page 83) for asynchronous resolution, you supply a reference to a structure of this type, along with a reference to a callback routine, defined by `AERemoteProcessResolverCallback` (page 148). The context is copied and the info pointer retained. When the callback is made, the info pointer is passed to the callback.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`AppleEvents.h`

ccntTokenRecord

Stores token information used by the AEResolve function while locating a range of objects.

```
struct ccntTokenRecord {
    DescType tokenClass;
    AEDesc token;
};
typedef struct ccntTokenRecord ccntTokenRecord;
```

Fields

`tokenClass`

The class ID of the container represented by the `token` parameter. See `DescType` (page 176).

`token`

A token for the current container. (Token is defined in `AEDisposeToken` (page 41). See `AEDesc` (page 162).

Discussion
When the `AEResolve` (page 89) function calls an object accessor function to locate a range of objects, the Apple Event Manager replaces the descriptor of type `typeCurrentContainer` with a token for the container of each boundary object. When using `AEResolve` to resolve the object specifier, your application doesn't need to examine the contents of this token, because the Apple Event Manager keeps track of it.

If your application attempts to resolve some or all of the object specifier without calling `AEResolve`, the application may need to examine the token before it can locate the boundary objects. The token provided by the Apple Event Manager for a boundary object's container is a descriptor of type `typeToken` whose data storage pointer refers to a structure of type `ccntTokenRecord`.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

IntlText

International text consists of an ordered series of bytes, beginning with a 4-byte language code and a 4-byte script code that together determine the format of the bytes that follow. (**Deprecated.** Use Unicode text instead.)

```
struct IntlText {
    ScriptCode theScriptCode;
    LangCode theLangCode;
    char theText[1];
};
typedef struct IntlText IntlText;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

OffsetArray

Specifies offsets of ranges of text. Not typically used by developers.

```
struct OffsetArray {
    sort fNumOfOffsets;
    long fOffset[1];
};
typedef struct OffsetArray OffsetArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

TextRange

Specifies a range of text. Not typically used by developers.

```
struct TextRange {
    long fStart;
    long fEnd;
    short fHiliteStyle;
};
typedef struct TextRange TextRange;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

TextRangeArray

Specifies an array of text ranges. Not typically used by developers.

```
struct TextRangeArray {
    short fNumOfRanges;
    TextRange fRange[1];
};
typedef struct TextRangeArray TextRangeArray;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

TScriptingSizeResource

Defines a data type to store stack and heap information. Not typically used by developers.

```
struct TScriptingSizeResource {
    short scriptingSizeFlags;
    unsigned long minStackSize;
    unsigned long preferredStackSize;
    unsigned long maxStackSize;
    unsigned long minHeapSize;
    unsigned long preferredHeapSize;
    unsigned long maxHeapSize;
};
typedef struct TScriptingSizeResource TScriptingSizeResource;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEUserTermTypes.h

WritingCode

```
struct WritingCode {
    ScriptCode theScriptCode;
    LangCode theLangCode;
};
typedef struct WritingCode WritingCode;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
AERegistry.h

AEAddressDesc

A descriptor that contains the address of an application. Typically used to describe the target application for an Apple event.

```
typedef AEDesc AEAddressDesc;
```

Discussion
An address descriptor is identical to a descriptor of data type AEDesc (page 162); however, the data for an address descriptor must always consist of the address of an application.

Every Apple event includes an attribute specifying the address of the target application. The address in an address descriptor can be specified as one of these types (or as any other descriptor type you define that can be coerced to one of these types): typeApplSignature, typeSessionID, or typeProcessSerialNumber. These constants are described in "Descriptor Type Constants" (page 197). You can also use "typeApplicationBundleID" (page 244).

If your application sends Apple events to itself using a typeProcessSerialNumber address descriptor with the lowLongOfPSN field set to kCurrentProcess (and the highLongOfPSN field set to 0), the Apple Event Manager jumps directly to the appropriate Apple event handler without going through the normal event-processing sequence.

Availability
Available in Mac OS X v10.0 and later.

Declared In
AEDataModel.h

AEArrayDataPointer

A pointer to a union of type AEArrayData.

```
typedef AEArrayData * AEArrayDataPointer
```

Discussion
This data type merely defines a pointer to an AEArrayData (page 161) union.

AEArrayType

Stores a value that specifies an array type.

```
typedef SInt8 AEArrayType;
```

Discussion
You use this data type with the `AEGetArray` (page 44) function and the `AEPutArray` (page 75) function to specify an array type, using one of the constants from "Data Array Constants" (page 196).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoerceDescUPP

Defines a data type for the universal procedure pointer for the `AECoerceDescProcPtr` callback function pointer.

```
typedef AECoerceDescProcPtr AECoerceDescUPP;
```

Discussion
For a description of a coerce descriptor callback function, see `AECoerceDescProcPtr` (page 140).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoercePtrUPP

Defines a data type for the universal procedure pointer for the `AECoercePtrProcPtr` callback function pointer.

```
typedef AECoercePtrProcPtr AECoercePtrUPP;
```

Discussion
For a description of a coerce pointer callback function, see `AECoercePtrProcPtr` (page 141).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AECoercionHandlerUPP

Defines a data type for the universal procedure pointer for the `AECoercionHandlerUPP` callback function pointer.

```
typedef AECoerceDescUPP AECoercionHandlerUPP;
```

Discussion
For a description of a coercion handler callback function, see `AECoercePtrProcPtr` (page 141).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDataStorage

A pointer to an opaque data type that provides storage for an `AEDesc` descriptor.

```
typedef AEStorageDataType * AEDataStorage;
```

Discussion
The Apple Event Manager defines the `AEDataStorage` data type to serve as a data storage field in the `AEDesc` (page 162) structure. Your application doesn't access the data pointed to by a data storage pointer directly. Rather, you work with the following functions:

- `AEGetDescDataSize` (page 50)
- `AEGetDescData` (page 48)
- `AEGetDescDataRange` (page 49)
- `AEReplaceDescData` (page 88)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDataStorageType

An opaque data type used to store data in Apple event descriptors.

```
typedef struct OpaqueAEDataStorageType * AEDataStorageType;
```

Discussion
See `AEDesc` (page 162) for related information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEDescList

A descriptor whose data consists of a list of one or more descriptors.

```
typedef AEDesc AEDescList;
```

Discussion
A descriptor list is identical to a descriptor of data type `AEDesc` (page 162) —the only difference is that the data in a descriptor list must always consist of a list of other descriptors.

Descriptor lists are a key building block of Apple events. Many Apple Event Manager functions take or return lists of descriptors in descriptor lists. For example, see the functions described in "Counting the Items in Descriptor Lists" (page 15) and "Getting Items From Descriptor Lists" (page 19).

The format of the data in the `dataHandle` of the descriptor is private. You can only operate on the contained elements with Apple Event Manager functions, including those described in "Counting the Items in Descriptor Lists" (page 15) and "Getting Items From Descriptor Lists" (page 19).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventSource

A data type for values that specify how an Apple event was delivered.

```
typedef SInt8 AEEventSource;
```

Discussion
"Event Source Constants" (page 204) lists the valid constant values for a variable or parameter of type `AEEventSource`.

You might use a variable of this type, for example, to get the source type of an Apple event by calling the function `AEGetAttributePtr` (page 46). You pass the `keyEventSourceAttr` constant as the value for the `theAEKeyWord` parameter and you pass a pointer to a variable of type `AEEventSource` for the `dataPtr` parameter. On return, the variable will contain one of the event source constant values described in "Event Source Constants" (page 204). The complete call looks like the following:

```
AppleEvent       theAppleEvent; // previously obtained Apple event
DescType        returnedType;
AEEventSource   sourceOfAE;
Size            actualSize;
OSErr           myErr;
myErr = AEGetAttributePtr(theAppleEvent,
                          keyEventSourceAttr,
                          typeShortInteger,
                          &returnedType,
                          (void *) &sourceOfAE,
                          sizeof (sourceOfAE),
                          &actualSize);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AppleEvents.h`

AEDisposeExternalUPP

Defines a universal procedure pointer to a function the Apple Event Manager calls to dispose of a descriptor created by the `AECreateDescFromExternalPtr` function.

```
typedef AEDisposeExternalProcPtr AEDisposeExternalUPP;
```

Discussion
See the `AEDisposeExternalProcPtr` (page 143) callback function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`AEDataModel.h`

AEEventClass

Specifies the event class of an Apple event.

```
typedef FourCharCode AEEventClass;
```

Discussion
Apple events are identified by their event class and event ID attributes. The event class is the attribute that identifies a group of related Apple events. When you call the `AEProcessAppleEvent` (page 73) function, the Apple Event Manager uses these attributes to identify a handler for a specific Apple event.

For more information on Apple event classes, see "Event Class Constants" (page 201).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventHandlerUPP

Defines a data type for the universal procedure pointer for the `AEEventHandlerUPP` callback function pointer.

```
typedef AEEventHandlerProcPtr AEEventHandlerUPP;
```

Discussion
For a description of an event handler callback function, see `AEEventHandlerProcPtr` (page 144).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEEventID

Specifies the event ID of an Apple event.

```
typedef FourCharCode AEEventID;
```

Discussion
Apple events are identified by their event class and event ID attributes. The event ID is the attribute that identifies a particular Apple event within its 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.

For more information on Apple event IDs, see "Event ID Constants" (page 202).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEFilterUPP

Defines a data type for the universal procedure pointer for the `AEFilterProcPtr` callback function pointer.

```
typedef AEFilterProcPtr AEFilterUPP;
```

Discussion
For a description of a filter callback function, see `AEFilterProcPtr` (page 146).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEIdleUPP

Defines a data type for the universal procedure pointer for the `AEIdleProcPtr` callback function pointer.

```
typedef AEIdleProcPtr AEIdleUPP;
```

Discussion
For a description of an idle callback function, see `AEIdleProcPtr` (page 147).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

AEKeyword

A four-character code that uniquely identifies a descriptor in an Apple event record or an Apple event.

```
typedef FourCharCode AEKeyword;
```

Discussion
The Apple Event Manager uniquely identifies the various parts of an Apple event by means of keywords associated with corresponding descriptors. Keywords are arbitrary names, stored as four-character codes of type `AEKeyword`. A keyword combined with a descriptor forms a keyword-specified descriptor, which is defined by a data structure of type `AERemoteProcessResolverContext` (page 163).

The Apple Event Manager also uses keywords for Apple event attributes. Keyword constants used by the Apple Event Manager are defined in "Keyword Attribute Constants" (page 209) and "Keyword Parameter Constants" (page 211).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERecord

A descriptor whose data is a list of keyword-specified descriptors.

```
typedef AEDescList AERecord;
```

Discussion
The Apple Event Manager provides routines that allow your application to create Apple event records and extract data from them when creating or responding to Apple events. You also work with Apple event records if your application resolves or creates object specifiers. Functions that use Apple event records are described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 18) and "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 14).

The descriptor list of keyword-specified descriptors in an Apple event record must specify Apple event parameters—they cannot specify Apple event attributes. Only descriptor lists of type Apple event can contain both attributes and parameters.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AERemoteProcessResolverRef

An opaque reference to an object that encapsulates the mechanism for obtaining a list of processes running on a remote machine.

```
typedef AERemoteProcessResolver * AERemoteProcessResolverRef;
```

Discussion
You create an instance of `AERemoteProcessResolverRef` by calling `AECreateRemoteProcessResolver` (page 36), and you must disposed of it by calling `AEDisposeRemoteProcessResolver` (page 40). An instance of this type is not a CFType (the base type used by all Core Foundation derived opaque types). For more information, see Core Foundation Reference Documentation.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AppleEvents.h`

AEReturnID

Specifies a return ID for a created Apple event.

```
typedef SInt16 AEReturnID;
```

Discussion
When you call the `AECreateAppleEvent` (page 32) function, you pass a value of type `AEReturnID` for the `returnID` parameter. "ID Constants for the AECreateAppleEvent Function" (page 205) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AESendOptions

This data type is not available. (**Deprecated.** Not available in Apple Event Manager API.)

```
typedef OptionBits AESendOptions;
```

AESendPriority

Specifies the processing priority for a sent Apple event.

```
typedef SInt16 AESendPriority;
```

Discussion
When you call the `AESend` (page 92) function, you pass a value of type `AESendPriority` for the `sendPriority` parameter. "Priority Constants for the AESend Function (Deprecated in Mac OS X)" (page 217) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AEStreamRef

An opaque data structure for storing stream-based descriptor data.

```
typedef struct OpaqueAEStreamRef * AEStreamRef;
```

Discussion
You create `AEStreamRef` objects and manipulate their contents using the "AEStream" routines found in the section "Creating Apple Event Structures Using Streams" (page 22)

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEHelpers.h`

AETransactionID

Specifies a transaction ID.

```
typedef SInt32 AETransactionID;
```

Discussion
A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. When you call the `AECreateAppleEvent` (page 32) function, you pass a value of type `AETransactionID` for the `transactionID` parameter. "ID Constants for the AECreateAppleEvent Function" (page 205) lists the valid constant values for a variable or parameter of this type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

AppleEvent

A descriptor whose data is a list of descriptors containing both attributes and parameters that make up an Apple event.

```
typedef AERecord AppleEvent;
```

Discussion
The Apple event data type describes a full-fledged Apple event. Like the data for an Apple event record (data type `AERecord` (page 173)), the data for an Apple event consists of a list of keyword-specified descriptors. Unlike an Apple event record, the data for an Apple event is conceptually divided into two parts, one for attributes and one for parameters. This division within the Apple event allows the Apple Event Manager to distinguish between an event's attributes and its parameters.

For additional information on the structure of an Apple event and on how to build one, see "Building an Apple Event" in Apple Events Programming Guide.

Many functions work with Apple events, including the functions described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 18), "Adding Parameters and Attributes to Apple Events and Apple Event Records" (page 14), "Creating an Apple Event" (page 15), and "Sending an Apple Event" (page 21).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

DescType

Specifies the type of the data stored in an `AEDesc` descriptor.

```
typedef ResType DescType;
```

Discussion
A `DescType` data type is a four-character code that stores a value that identifies the data in an `AEDesc` (page 162) descriptor, the basic building block for all Apple events.

The descriptor type constants used by the Apple Event Manager are described in "Descriptor Type Constants" (page 197) and "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEDataModel.h`

OffsetArrayHandle

Defines a data type that points to an `OffsetArray`. Not typically used by developers.

```
typedef OffsetArrayPtr * OffsetArrayHandle;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AERegistry.h`

OSLAccessorUPP

Defines a data type for the universal procedure pointer for the `OSLAccessorProcPtr` callback function pointer.

```
typedef OSLAccessorProcPtr OSLAccessorUPP;
```

Discussion
For a description of an object accessor callback function, see `OSLAccessorProcPtr` (page 149).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLAdjustMarksUPP

Defines a data type for the universal procedure pointer for the `OSLAdjustMarksProcPtr` callback function pointer.

```
typedef OSLAdjustMarksProcPtr OSLAdjustMarksUPP;
```

Discussion
For a description of an adjust marks callback function, see `OSLAdjustMarksProcPtr` (page 151).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCompareUPP

Defines a data type for the universal procedure pointer for the `OSLCompareProcPtr` callback function pointer.

```
typedef OSLCompareProcPtr OSLCompareUPP;
```

Discussion
For a description of a compare callback function, see `OSLCompareProcPtr` (page 152).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLCountUPP

Defines a data type for the universal procedure pointer for the `OSLCountProcPtr` callback function pointer.

```
typedef OSLCountProcPtr OSLCountUPP;
```

Discussion
For a description of a count callback function, see `OSLCountProcPtr` (page 154).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLDisposeTokenUPP

Defines a data type for the universal procedure pointer for the `OSLDisposeTokenProcPtr` callback function pointer.

```
typedef OSLDisposeTokenProcPtr OSLDisposeTokenUPP;
```

Discussion
For a description of a dispose token callback function, see `OSLDisposeTokenProcPtr` (page 155).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetErrDescUPP

Defines a data type for the universal procedure pointer for the `OSLGetErrDescProcPtr` callback function pointer.

```
typedef OSLGetErrDescProcPtr OSLGetErrDescUPP;
```

Discussion
For a description of a get error descriptor callback function, see `OSLGetErrDescProcPtr` (page 157).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLGetMarkTokenUPP

Defines a data type for the universal procedure pointer for the `OSLGetMarkTokenProcPtr` callback function pointer.

```
typedef OSLGetMarkTokenProcPtr OSLGetMarkTokenUPP;
```

Discussion
For a description of a mark token callback function, see `OSLGetMarkTokenProcPtr` (page 158).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

OSLMarkUPP

Defines a data type for the universal procedure pointer for the `OSLMarkProcPtr` callback function pointer.

```
typedef OSLMarkProcPtr OSLMarkUPP;
```

Discussion
For a description of a mark callback function, see `OSLMarkProcPtr` (page 160).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEObjects.h`

AEInteractAllowed

Specifies an interaction level.

```
typedef SInt8 AEInteractAllowed;
```

Discussion
When you call the `AEGetInteractionAllowed` (page 52) function or the `AESetInteractionAllowed` (page 95) function, you receive or pass a value of type `AEInteractAllowed` for the `level` parameter. Interaction levels are described and the valid interaction level constants are listed in "User Interaction Level Constants" (page 221).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`AEInteraction.h`

Constants

AEBuild Error Codes

Represents syntax errors found by an "AEBuild" routine.

```
typedef UInt32 AEBuildErrorCode;
enum {
aeBuildSyntaxNoErr = 0,
aeBuildSyntaxBadToken = 1,
aeBuildSyntaxBadEOF = 2,
aeBuildSyntaxNoEOF = 3,
aeBuildSyntaxBadNegative = 4,
aeBuildSyntaxMissingQuote = 5,
aeBuildSyntaxBadHex = 6,
aeBuildSyntaxOddHex = 7,
aeBuildSyntaxNoCloseHex = 8,
aeBuildSyntaxUncoercedHex = 9,
aeBuildSyntaxNoCloseString = 10,
aeBuildSyntaxBadDesc = 11,
aeBuildSyntaxBadData = 12,
aeBuildSyntaxNoCloseParen = 13,
aeBuildSyntaxNoCloseBracket = 14,
aeBuildSyntaxNoCloseBrace = 15,
aeBuildSyntaxNoKey = 16,
aeBuildSyntaxNoColon = 17,
aeBuildSyntaxCoercedList = 18,
aeBuildSyntaxUncoercedDoubleAt = 19
};
```

Constants

`aeBuildSyntaxNoErr`

No error.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadToken`

An illegal character was specified.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadEOF`

An unexpected end of format string was encountered.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxNoEOF`

There were unexpected characters beyond the end of the format string.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadNegative`

A minus sign "-" was not followed by digits.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxMissingQuote`

A string was not terminated by a closing quotation mark.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxBadHex`
A hex string contained characters other than hexadecimal digits.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxOddHex`
A hex string contained an odd number of digits.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseHex`
A hex string was missing a "$" or "»" character.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxUncoercedHex`
A hex string must be coerced to a type.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseString`
A string was missing a closing quote.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxBadDesc`
An illegal descriptor was specified.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxBadData`
Bad data was found inside a variable argument list.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseParen`
A data value was missing a closing parenthesis.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseBracket`
A comma or closing bracket "]" was expected.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoCloseBrace`
A comma or closing brace "}" was expected.
Available in Mac OS X v10.0 and later.
Declared in `AEHelpers.h`.

`aeBuildSyntaxNoKey`
A keyword was missing from a descriptor.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxNoColon`
In a descriptor, one of the keywords was not followed by a colon.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxCoercedList`
Cannot coerce a list.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

`aeBuildSyntaxUncoercedDoubleAt`
You must coerce a "@@" substitution.

Available in Mac OS X v10.0 and later.

Declared in `AEHelpers.h`.

AESendMode

Specify send preferences to the `AESend` function.

```
typedef SInt32 AESendMode;
enum {
    kAENoReply = 0x00000001,
    kAEQueueReply = 0x00000002,
    kAEWaitReply = 0x00000003,
    kAEDontReconnect = 0x00000080,
    kAEWantReceipt = 0x00000200,
    kAENeverInteract = 0x00000010,
    kAECanInteract = 0x00000020,
    kAEAlwaysInteract = 0x00000030,
    kAECanSwitchLayer = 0x00000040,
    kAEDontRecord = 0x00001000,
    kAEDontExecute = 0x00002000,
    kAEProcessNonReplyEvents = 0x00008000
};
```

Constants

`kAENoReply`
The reply preference—your application does not want a reply Apple event. If you set the bit specified by this constant, the server processes the Apple event as soon as it has the opportunity.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEQueueReply`

The reply preference—your application wants a reply Apple event. If you set the bit specified by this constant, the reply appears in your event queue as soon as the server has the opportunity to process and respond to your Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEWaitReply`

The reply preference—your application wants a reply Apple event and is willing to give up the processor while waiting for the reply. For example, if the server application is on the same computer as your application, your application yields the processor to allow the server to respond to your Apple event.

If you set the bit specified by this constant, you must provide an idle function. This function should process any update events, null events, operating-system events, or activate events that occur while your application is waiting for a reply. For more information on idle routines, see `AEInteractWithUser` (page 69).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontReconnect`

Deprecated and unsupported in Mac OS X. The reconnection preference—the Apple Event Manager must not automatically try to reconnect if it receives a `sessClosedErr` result code from the PPC Toolbox. If you don't set this flag, the Apple Event Manager automatically attempts to reconnect and reestablish the session.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEWantReceipt`

Deprecated and unsupported in Mac OS X. The return receipt preference—the sender wants to receive a return receipt for this Apple event from the Event Manager. (A return receipt means only that the receiving application accepted the Apple event the Apple event may or may not be handled successfully after it is accepted.) If the receiving application does not send a return receipt before the request times out, `AESend` returns `errAETimeout` as its function result.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAENeverInteract`

The user interaction preference—the server application should never interact with the user in response to the Apple event. If you set the bit specified by this constant, the `AEInteractWithUser` (page 69) function (when called by the server) returns the `errAENoUserInteraction` result code. When you send an Apple event to a remote application, the default is to set this bit.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAECanInteract`

The user interaction preference—the server application can interact with the user in response to the Apple event. By convention, you set the bit specified by this constant if the user needs to supply information to the server. If you set the bit and the server allows interaction, the `AEInteractWithUser` (page 69) function either brings the server application to the foreground or posts a notification request. When you send an Apple event to a local application, the default is to set this bit.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEAlwaysInteract`

The user interaction preference—the server application should always interact with the user in response to the Apple event. By convention, you set the bit specified by this constant whenever the server application normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user. If you set the bit specified by this constant, the `AEInteractWithUser` (page 69) function either brings the server application to the foreground or posts a notification request.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAECanSwitchLayer`

The application switch preference—if both the client and server allow interaction, and if the client application is the active application on the local computer and is waiting for a reply (that is, it has set the `kAEWaitReply` flag), `AEInteractWithUser` brings the server directly to the foreground. Otherwise, `AEInteractWithUser` uses the Notification Manager to request that the user bring the server application to the foreground.

You should specify the `kAECanSwitchLayer` flag only when the client and server applications reside on the same computer. In general, you should not set this flag if it would be confusing or inconvenient to the user for the server application to come to the front unexpectedly. This flag is ignored if you are sending an Apple event to a remote computer.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontRecord`

The recording preference—your application is sending an event to itself but does not want the event recorded. When Apple event recording is on, the Apple Event Manager records a copy of every event your application sends to itself except for those events for which this flag is set.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDontExecute`

The execution preference—your application is sending an Apple event to itself for recording purposes only—that is, you want the Apple Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEProcessNonReplyEvents`

Allow processing of non-reply Apple events while awaiting a synchronous Apple event reply (you specified `kAEWaitReply` for the reply preference).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
You use these constants with the `sendMode` parameter to the `AESend` (page 92) function to specify how the server application should handle the reply mode, the interaction level, the application switch mode, the reconnection mode, the return receipt mode, the recording mode, and whether to process non-reply Apple events. To obtain a value for this parameter, you add together constants to set the appropriate bits for the Apple event you are about to send. The following paragraphs provide additional information about how you use these constants.

You can set only one flag reply preference (`kAENoReply`, `kAEQueueReply`, or `kAEWaitReply`), one user interaction preference (`kAENeverInteract`, `kAECanInteract`, or `kAEAlwaysInteract`), and one recording and execution preference (`kAEDontRecord` or `kAEDontExecute`).

Before the Apple Event Manager sends a reply event back to the client application, the `keyAddressAttr` attribute contains the address of the client application. After the client receives the reply event, the `keyAddressAttr` attribute contains the address of the server application.

If you specify `kAEWaitReply`, the Apple Event Manager uses the Event Manager to send the event. The Apple Event Manager then calls the `WaitNextEvent` function on behalf of your application, causing your application to yield the processor and giving the server application a chance to receive and handle the Apple event. Your application continues to yield the processor until the server handles the Apple event or the request times out.

Specify the `kAEWantReceipt` flag if your application wants notification that the server application has accepted the Apple event. If you specify this flag, your application receives a return receipt as a high-level event.

If you specify the `kAEWantReceipt` flag and the server application does not accept the Apple event within the time specified by the `timeOutInTicks` parameter to `AESend`, the `AESend` function returns a timeout error. Note that `AESend` also returns a timeout error if your application sets the `kAEWaitReply` flag and does not receive the reply Apple event within the time specified by the `timeOutInTicks` parameter.

You use one of the three flags—`kAENeverInteract`, `kAECanInteract`, and `kAEAlwaysInteract`—to specify whether the server should interact with the user when handling the Apple event. Specify `kAENeverInteract` if the server should not interact with the user when handling the Apple event. You might specify this constant if you don't want the user to be interrupted while the server is handling the Apple event.

Use the `kAECanInteract` flag if the server should interact with the user when the user needs to supply information to the server. Use the `kAEAlwaysInteract` flag if the server should interact with the user whenever the server normally asks a user to confirm a decision or interact in any other way, even if no additional information is needed from the user. Note that it is the responsibility of the server and client applications to agree on how to interpret the `kAEAlwaysInteract` flag.

If the client application does not set any one of the user interaction flags, the Apple Event Manager sets a default, depending on the location of the target of the Apple event. If the server application is on a remote computer, the Apple Event Manager sets the `kAENeverInteract` flag as the default. If the target of the Apple event is on the local computer, the Apple Event Manager sets the `kAECanInteract` flag as the default.

The server application should call `AEInteractWithUser` if it needs to interact with the user. If both the client and the server allow user interaction, the Apple Event Manager attempts to bring the server to the foreground if it is not already the foreground process. If both the `kAECanSwitchLayer` and the `kAEWaitReply` flags are set, and if the client application is the active application on the local computer, the Apple Event Manager brings the server application directly to the front. Otherwise, the Apple Event Manager

posts a notification request asking the user to bring the server application to the front, regardless of whether the `kAECanSwitchLayer` flag is set. This ensures that the user will not be interrupted by an unexpected application switch.

Specify the `kAEDontRecord` flag if your application is sending an Apple event to itself that you don't want to be recorded. When Apple event recording has been turned on, every event that your application sends to itself will be automatically recorded by the Apple Event Manager except those sent with the `kAEDontRecord` flag set.

Specify the `kAEDontExecute` flag if your application is sending an Apple event to itself for recording purposes only—that is, if you want the Apple Event Manager to send a copy of the event to the recording process but you do not want your application actually to receive the event.

See also "Requesting User Interaction" (page 21).

Version Notes
The `kAEDontReconnect` and `kAEWantReceipt` constants are deprecated and unsupported in Mac OS X.

Declared In
`AEDataModel.h`

Apple Event Recording Event ID Constants

Specify event IDs for events that deal with Apple event recording.

```
enum {
    kAEStartRecording = 'reca',
    kAEStopRecording = 'recc',
    kAENotifyStartRecording = 'rec1',
    kAENotifyStopRecording = 'rec0',
    kAENotifyRecording = 'recr'
};
```

Constants

`kAEStartRecording`

Event ID for an event by a scripting component to the recording process (or to any running process on the local computer), but handled by the Apple Event Manager. The Apple Event Manager responds by turning on recording and sending a `recording on` event to all running processes on the local computer.

If sent by process serial number (PSN), this event must be addressed using a real PSN; it should never be sent to an address specified as `kCurrentProcess`.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEStopRecording`

Event ID for an event sent by a scripting component to the recording process (or to any running process on the local computer), but handled by the Apple Event Manager. The Apple Event Manager responds by sending a `recording off` event to all running processes on the local computer.

If sent by a PSN, this event must be addressed using a real PSN; it should never be sent to an address specified as `kCurrentProcess`.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAENotifyStartRecording`
: An event that notifies an application that recording has been turned on.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAENotifyStopRecording`
: An event that notifies an application that recording has been turned off.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAENotifyRecording`
: Wildcard event class and event ID handled by a recording process in order to receive and record copies of recordable events sent to it by the Apple Event Manager. Scripting components install a handler for this event on behalf of a recording process when recording is turned on and remove the handler when recording is turned off.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Version Notes
These constants are available only in version 1.0.1 and later of the Apple Event Manager.

cAEList

```
enum {
    cAEList = 'list',
    cApplication = 'capp',
    cArc = 'carc',
    cBoolean = 'bool',
    cCell = 'ccel',
    cChar = 'cha ',
    cColorTable = 'clrt',
    cColumn = 'ccol',
    cDocument = 'docu',
    cDrawingArea = 'cdrw',
    cEnumeration = 'enum',
    cFile = 'file',
    cFixed = 'fixd',
    cFixedPoint = 'fpnt',
    cFixedRectangle = 'frct',
    cGraphicLine = 'glin',
    cGraphicObject = 'cgob',
    cGraphicShape = 'cgsh',
    cGraphicText = 'cgtx',
    cGroupedGraphic = 'cpic'
};
```

Callback Constants for the AEResolve Function

Specify supported callback features to the `AEResolve` function.

```
enum {
    kAEIDoMinimum = 0x0000,
    kAEIDoWhose = 0x0001,
    kAEIDoMarking = 0x0004,
    kAEPassSubDescs = 0x0008,
    kAEResolveNestedLists = 0x0010,
    kAEHandleSimpleRanges = 0x0020,
    kAEUseRelativeIterators = 0x0040
};
```

Constants

`kAEIDoMinimum`

The application does not handle whose tests or provide marking callbacks.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`kAEIDoWhose`

The application supports whose tests (supports key form `formWhose`).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`kAEIDoMarking`

The application provides marking callback functions. Marking callback functions are described in "Object Callback Functions" (page 140).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion

You use these constants to supply a value for the `callbackFlags` parameter to the `AEResolve` (page 89) function. This value specifies whether your application supports whose descriptors or provides marking callback functions. To obtain a value for this parameter, you can add together constants to set the appropriate bits, as shown in the following example (for an application that supports both whose tests and marking):

```
    AEDesc objectSpecifier; // Previously obtained object specifier.     AEDesc
  resultToken;
    OSErr myErr;

    myErr = AEResolve (&objectSpecifier,
                       kAEIDoWhose + kAEIDoMarking, &resultToken)
```

AppleScript generates whose clauses from script statements such as the following:

```
tell application "Finder"
    every file in control panels folder whose file type is "APPL"
end tell
```

cInsertionLoc

```
enum {
    cInsertionLoc = 'insl',
    cInsertionPoint = 'cins',
    cIntlText = 'itxt',
    cIntlWritingCode = 'intl',
    cItem = 'citm',
    cLine = 'clin',
    cLongDateTime = 'ldt ',
    cLongFixed = 'lfxd',
    cLongFixedPoint = 'lfpt',
    cLongFixedRectangle = 'lfrc',
    cLongInteger = 'long',
    cLongPoint = 'lpnt',
    cLongRectangle = 'lrct',
    cMachineLoc = 'mLoc',
    cMenu = 'cmnu',
    cMenuItem = 'cmen',
    cObject = 'cobj',
    cObjectSpecifier = 'obj ',
    cOpenableObject = 'coob',
    cOval = 'covl'
};
```

cKeystroke

```
enum {
    cKeystroke = 'kprs',
    pKeystrokeKey = 'kMsg',
    pModifiers = 'kMod',
    pKeyKind = 'kknd',
    eModifiers = 'eMds',
    eOptionDown = 'Kopt',
   eCommandDown = 'Kcmd',
    eControlDown = 'Kctl',
    eShiftDown = 'Ksft',
    eCapsLockDown = 'Kclk',
    eKeyKind = 'ekst',
    eEscapeKey = 0x6B733500,
    eDeleteKey = 0x6B733300,
    eTabKey = 0x6B733000,
    eReturnKey = 0x6B732400,
    eClearKey = 0x6B734700,
    eEnterKey = 0x6B734C00,
    eUpArrowKey = 0x6B737E00,
    eDownArrowKey = 0x6B737D00,
    eLeftArrowKey = 0x6B737B00,
    eRightArrowKey = 0x6B737C00,
    eHelpKey = 0x6B737200,
    eHomeKey = 0x6B737300,
    ePageUpKey = 0x6B737400,
    ePageDownKey = 0x6B737900,
    eForwardDelKey = 0x6B737500,
    eEndKey = 0x6B737700,
    eF1Key = 0x6B737A00,
```

```
    eF2Key = 0x6B737800,
    eF3Key = 0x6B736300,
    eF4Key = 0x6B737600,
    eF5Key = 0x6B736000,
    eF6Key = 0x6B736100,
    eF7Key = 0x6B736200,
    eF8Key = 0x6B736400,
    eF9Key = 0x6B736500,
    eF10Key = 0x6B736D00,
    eF11Key = 0x6B736700,
    eF12Key = 0x6B736F00,
    eF13Key = 0x6B736900,
    eF14Key = 0x6B736B00,
    eF15Key = 0x6B737100
};
```

Comparison Operator Constants

Specify a comparison operation to perform on two operands.

```
enum {
    kAEAsk = 'ask ',
    kAEBefore = 'befo',
    kAEBeginning = 'bgng',
    kAEBeginsWith = 'bgwt',
    kAEBeginTransaction = 'begi',
    kAEBold = 'bold',
    kAECaseSensEquals = 'cseq',
    kAECentered = 'cent',
    kAEChangeView = 'view',
    kAEClone = 'clon',
    kAEClose = 'clos',
    kAECondensed = 'cond',
    kAEContains = 'cont',
    kAECopy = 'copy',
    kAECoreSuite = 'core',
    kAECountElements = 'cnte',
    kAECreateElement = 'crel',
    kAECreatePublisher = 'cpub',
    kAECut = 'cut ',
    kAEDelete = 'delo'
};
```

Constants

`kAEBeginsWith`

The value of `operand1` begins with the value of `operand2` (for example, the string `"operand"` begins with the string `"opera"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEContains`

The value of `operand1` contains the value of `operand2` (for example, the string `"operand"` contains the string `"era"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAECoreSuite`
: An Apple event in the Standard Suite.

  Available in Mac OS X v10.0 and later.

  Declared in `AERegistry.h`.

Discussion
When you call the `CreateCompDescriptor` (page 114) function, you pass one of these comparison operators in the `comparisonOperator` parameter. The `CreateCompDescriptor` function creates a comparison descriptor that specifies how to compare one or more Apple event objects with either another Apple event object or a descriptor.

The actual comparison of the two operands is performed by the object comparison function provided by the client application—see `OSLCompareProcPtr` (page 152). The way a comparison operator is interpreted is up to each application.

For related information, see "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191).

Constants for Object Specifiers, Positions, and Logical and Comparison Operations

Specify the types of the four keyword-specified descriptors that make up the data in an object specifier, as well as constants for position, logical operations, and comparison operations.

```
enum {
    kAEAND = 'AND ',
    kAEOR = 'OR  ',
    kAENOT = 'NOT ',
    kAEFirst = 'firs',
    kAELast = 'last',
    kAEMiddle = 'midd',
    kAEAny = 'any ',
    kAEAll = 'all ',
    kAENext = 'next',
    kAEPrevious = 'prev',
    keyAECompOperator = 'relo',
    keyAELogicalTerms = 'term',
    keyAELogicalOperator = 'logc',
    keyAEObject1 = 'obj1',
    keyAEObject2 = 'obj2',
    keyAEDesiredClass = 'want',
    keyAEContainer = 'from',
    keyAEKeyForm = 'form',
    keyAEKeyData = 'seld'
};
```

Constants
`kAEAND`
: Specifies a logical `AND` operation.

  Available in Mac OS X v10.0 and later.

  Declared in `AEObjects.h`.

`kAEOR`
Specifies a logical `OR` operation.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAENOT`
Specifies a logical `NOT` operation.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEFirst`
The first element in the specified container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAELast`
Specifies the last element in the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEMiddle`
Specifies the middle element in the container. If an object specifier specifies `kAEMiddle` and the number of elements in the container is even, the Apple Event Manager rounds down. For example, in a range of four words the second word is the "middle" word.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEAny`
Specifies a single element chosen at random from the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEAll`
Specifies all the elements in the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAENext`
Specifies the Apple event object after the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`kAEPrevious`
Specifies the Apple event object before the container.
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAECompOperator`
Specifies a descriptor of `typeType`, whose data consists of one of the constant values described in "Key Form and Descriptor Type Object Specifier Constants" (page 206).
Available in Mac OS X v10.0 and later.
Declared in `AEObjects.h`.

`keyAELogicalTerms`
Specifies a descriptor of type `typeAEList` containing one or more comparison or logical descriptors.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAELogicalOperator`
Specifies a descriptor of type `typeEnumerated` whose data is one of the logical operators (such as `kAEAND`) defined in "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEObject1`
Identifies a descriptor for the element that is currently being compared to the object or data specified by the descriptor for the keyword `keyAEObject2`. Either object can be described by a descriptor of type `typeObjectSpecifier` or `typeObjectBeingExamined`.

A descriptor of `typeObjectBeingExamined` acts as a placeholder for each of the successive elements in a container when the Apple Event Manager tests those elements one at a time.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEObject2`
Identifies a descriptor for the element that is currently being compared to the object or data specified by the descriptor for the keyword `keyAEObject1`.

The keyword `keyAEObject2` can also be used with a descriptor of any other descriptor type whose data is to be compared to each element in a container.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEDesiredClass`
A four-character code that identifies the object class of the specified object or objects.

Constants for object class IDs are described in "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEContainer`
Specifies the container for the requested object or objects. The data is an object specifier (or in some cases a null descriptor).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEKeyForm`
A four-character code that identifies the key form for the specified object or objects.

The constants for specifying the key form are described in "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEKeyData`

Data or nested descriptors that specify a property, name, position, range, or test, depending on the key form.

The descriptor types used in object specifiers are described in "Key Form and Descriptor Type Object Specifier Constants" (page 206).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion

When you call the `CreateLogicalDescriptor` (page 115) function to create a logical descriptor, you pass one of the logical operators `kAEAND`, `kAEOR`, or `kAENOT` in the `theLogicOperator` parameter. The `CreateLogicalDescriptor` function creates a logical descriptor that specifies a logical operation to perform on one or more operands.

The constants `kAEFirst`, `kAELast`, `kAEMiddle`, `kAEAny`, and `kAEAll` provide the key data for a keyword-specified descriptor of key form `formAbsolutePosition` and descriptor type `typeAbsoluteOrdinal`.

The constants `kAENext`, and `kAEPrevious` provide the key data for a keyword-specified descriptor of key form `formRelativePosition`.

Key form constants and descriptor type constants for object specifiers are defined in "Key Form and Descriptor Type Object Specifier Constants" (page 206).

The constants `keyAELogicalTerms` and `keyAELogicalOperator` define the keyword descriptors for a logical descriptor. A logical descriptor is a coerced Apple event record of type `typeLogicalDescriptor` that specifies a logical expression—that is, an expression that the Apple Event Manager evaluates to either `TRUE` or `FALSE`. You can create a logical descriptor with the `CreateLogicalDescriptor` (page 115) function.

The data for a logical descriptor consists of two keyword-specified descriptors: the first with descriptor type `keyAELogicalOperator`, descriptor type `typeEnumerated`, and one of the logical operators defined in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191) for its data; and the second with descriptor type `keyAELogicalTerms`, descriptor type `typeEnumerated`, and one or more comparison or logical descriptors for its data. Comparison constants are described in "Comparison Operator Constants" (page 190).

The logical expression is constructed from a logical operator (one of the Boolean operators `AND`, `OR`, or `NOT`) and a list of logical terms to which the operator is applied (where `NOT` can only be used where the list of terms is a single-item list). Each logical term in the list can be either another logical descriptor or a comparison descriptor (described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191)).

The Apple Event Manager short-circuits its evaluation of a logical expression as soon as one part of the expression fails a test. For example, if while testing a logical expression such as A `AND` B `AND` C the Apple Event Manager discovers that A `AND` B is not true, it will evaluate the expression to `FALSE` without testing C.

The constants `keyAECompOperator`, `keyAEObject1`, and `keyAEObject2` define the keyword descriptors for a comparison descriptor. A comparison descriptor is a coerced Apple event record of type `typeCompDescriptor` that specifies an Apple event object and either another Apple event object or data for the Apple Event Manager to compare to the first object. You can create a logical descriptor with the `CreateCompDescriptor` (page 114) function.

The Apple Event Manager can also use the information in a comparison descriptor to compare elements in a container, one at a time, either to an Apple event object or to data. The data for a comparison descriptor consists of three keyword-specified descriptors:

- A descriptor with keyword `keyAECompOperator`, descriptor type `typeType`, and one of the logical operators defined in "Comparison Operator Constants" (page 190) for its data.
- A descriptor with keyword `keyAEObject1` and either
- descriptor type `typeObjectSpecifier` and object specifier data to compare, or
- descriptor type `typeObjectBeingExamined` and a data storage pointer of `NULL`.
- A descriptor with keyword `keyAEObject2` and either
- descriptor type `typeObjectSpecifier` and object specifier data to compare, or
- descriptor type `typeObjectBeingExamined` and a data storage pointer of `NULL`, or
- any other descriptor type and the data to be compared for that descriptor type.

You don't have to support all the available comparison operators for all Apple event objects for example, the `begins with` operator probably doesn't make sense for objects of type `cRectangle`. It is up to you to decide which comparison operators are appropriate for your application to support, and how to interpret them. If necessary, you can define your own custom comparison operators. If you think you need to do this, check the Apple Events and Scripting header files to see if existing definitions of comparison operators can be adapted to the needs of your application.

An object specifier is a coerced Apple event record of descriptor type `typeObjectSpecifier` whose data contains consists of four keyword-specified descriptors. The constants `keyAEDesiredClass`, `keyAEContainer`, `keyAEKeyForm`, and `keyAEKeyData` specify the keywords for the four descriptor types that together identify the specified object or objects.

cURL

```
enum {
    cURL = 'url ',
    cInternetAddress = 'IPAD',
    cHTML = 'html',
    cFTPItem = 'ftp '
};
```

Constants

`cURL`

Specifies a Uniform Resource Locator or Uniform Resource ID (URI).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cInternetAddress`

Specifies an Internet or Intranet address for the TCP/IP protocol.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cHTML`
Specifies HTML (HyperText Markup Language) format.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cFTPItem`
Specifies FTP (File Transfer Protocol) protocol.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

cVersion

```
enum {
    cVersion = 'vers',
    cWindow = 'cwin',
    cWord = 'cwor',
    enumArrows = 'arro',
    enumJustification = 'just',
    enumKeyForm = 'kfrm',
    enumPosition = 'posi',
    enumProtection = 'prtn',
    enumQuality = 'qual',
    enumSaveOptions = 'savo',
    enumStyle = 'styl',
    enumTransferMode = 'tran',
    formUniqueID = 'ID  ',
    kAEAbout = 'abou',
    kAEAfter = 'afte',
    kAEAliasSelection = 'sali',
    kAEAllCaps = 'alcp',
    kAEArrowAtEnd = 'aren',
    kAEArrowAtStart = 'arst',
    kAEArrowBothEnds = 'arbo'
};
```

Constants

`formUniqueID`
Specifies a value that uniquely identifies an object within its container or across an application.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Data Array Constants

Specify an array type for storing or extracting descriptor lists with the `AEPutArray` and `AEGetArray` functions.

```
enum {
    kAEDataArray = 0,
    kAEPackedArray = 1,
    kAEDescArray = 3,
    kAEKeyDescArray = 4
};
```

Constants

`kAEDataArray`

Array items consist of data of the same size and same type, and are aligned on word boundaries.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEPackedArray`

Array items consist of data of the same size and same type, and are packed without regard for word boundaries.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEDescArray`

Array items consist of descriptors of different descriptor types with data of variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEKeyDescArray`

Array items consist of keyword-specified descriptors with different keywords, different descriptor types, and data of variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

When your application calls the `AEPutArray` (page 75) function to put information into a descriptor list or the `AEGetArray` (page 44) function to get information from a descriptor list, it uses an array to store the information. The type of array depends on the data for the array, as specified by one of these constants.

Array items in Apple event arrays of type `kAEDataArray`, `kAEPackedArray`, or `kAEHandleArray` must be factored—that is, contained in a factored descriptor list. For more information, see `AEPutArray` (page 75).

Descriptor Type Constants

Specify types for descriptors.

```
enum {
    typeAEList = 'list',
    typeAERecord = 'reco',
    typeAppleEvent = 'aevt',
    typeEventRecord = 'evrc',
    typeTrue = 'true',
    typeFalse = 'fals',
    typeAlias = 'alis',
    typeEnumerated = 'enum',
    typeType = 'type',
    typeAppParameters = 'appa',
    typeProperty = 'prop',
    typeFSS = 'fss ',
    typeFSRef = 'fsrf',
    typeFileURL = 'furl',
    typeKeyword = 'keyw',
    typeSectionH = 'sect',
    typeWildCard = '****',
    typeApplSignature = 'sign',
    typeQDRectangle = 'qdrt',
    typeFixed = 'fixd',
    typeProcessSerialNumber = 'psn ',
    typeApplicationURL = 'aprl',
    typeNull = 'null'
};
```

Constants

`typeAEList`
List of descriptors.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeAERecord`
List of keyword-specified descriptors.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeAppleEvent`
Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeTrue`
`TRUE` Boolean value.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeFalse`
`FALSE` Boolean value.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeAlias`
Alias.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeEnumerated`
Enumerated data.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeType`
Four-character code for event class or event ID

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeAppParameters`
Process Manager launch parameters.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeProperty`
Apple event object property.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeFSS`
File system specification. Deprecated in Mac OS X. Use file system references (`typeFSRef`) instead.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeFSRef`
File system reference. Use in preference to file system specifications (`typeFSS`).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeFileURL`
A file URL. That is, the associated data consists of the bytes of a UTF-8 encoded URL with a scheme of "file". This type is appropriate for describing a file that may not yet exist—see Technical Note 2022 for more information.

You can translate between a descriptor of this type and an instance of `CFURL` by calling `CFURLCreateWithBytes` and specifying `kCFStringEncodingUTF8` for the encoding. Or, if you have a `CFURLRef`, you can call `CFURLCreateData` to get the data as an instance of `CFData` (again specifying an encoding of `kCFStringEncodingUTF8`), and`CFDataGetBytes` to get the actual bytes to insert into a descriptor of this type.

Available in Mac OS X v10.1 and later.

Declared in `AEDataModel.h`.

`typeKeyword`
Apple event keyword.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeSectionH`
Handle to a section record. (Deprecated.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeWildCard`
Matches any type.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeApplSignature`
Application signature.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeProcessSerialNumber`
A process serial number. See also `AEAddressDesc` (page 167).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeApplicationURL`
For specifying an application by URL. See Discussion section below for important information.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeNull`
A null data storage pointer. When resolving an object specifier, an object with a null storage pointer specifies the default container at the top of the container hierarchy.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
The constants described here specify the data type for a descriptor and show the kind of data stored in a descriptor with that type.

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 162), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 176). AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Numeric Descriptor Type Constants" (page 213) and "Other Descriptor Type Constants" (page 217). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

For the constant `typeApplicationURL`, the data that specifies the application URL takes the following format:

```
eppc://[username[:password]@]host/AppName[[?uid=#]&[pid=#]]
```

As indicated by this format:

- `username` is optional. If present, an `'@'` must appear before the host name. `password` is optional. If present, `username` is not optional, and the password must be separated from the username by a `':'` and must precede the `'@'`. `AppName` is not optional; if it contains non-UTF-8 characters or white space, it must be URL-encoded (for example, `My%20Application`).

- `uid` and `pid` are optional. If `pid` is present, `uid` and `AppName` are ignored and the event is delivered only to applications with the given process id. If `uid` is present, events are directed to the application name owned by the given user id.

The following are examples of valid URLs:

```
eppc://Steve%20Zellers:wombat@grrr.apple.com/Microsoft%20Word
eppc://Steve%20Zellers:wombat@grrr.apple.com/Microsoft%20Word?pid=1284
```

The availability of user identifiers provides enhanced Apple event support for Fast User Switching. Such identifiers make it possible to send Apple events to applications running in any session, if the uids of the processes match. `'root'` (or `uid 0`) processes are allowed to send Apple events to any process in any session. Non-root processes can only target applications that match their uid.

eScheme

```
enum {
    eScheme = 'esch',
    eurlHTTP = 'http',
    eurlHTTPS = 'htps',
    eurlFTP = 'ftp ',
    eurlMail = 'mail',
    eurlFile = 'file',
    eurlGopher = 'gphr',
    eurlTelnet = 'tlnt',
    eurlNews = 'news',
    eurlSNews = 'snws',
    eurlNNTP = 'nntp',
    eurlMessage = 'mess',
    eurlMailbox = 'mbox',
    eurlMulti = 'mult',
    eurlLaunch = 'laun',
    eurlAFP = 'afp ',
    eurlAT = 'at  ',
    eurlEPPC = 'eppc',
    eurlRTSP = 'rtsp',
    eurlIMAP = 'imap',
    eurlNFS = 'unfs',
    eurlPOP = 'upop',
    eurlLDAP = 'uldp',
    eurlUnknown = 'url?'
};
```

Event Class Constants

Specify the event class for an Apple event.

```
enum {
    kCoreEventClass = 'aevt'
};
```

Constants

`kCoreEventClass`

An Apple event sent by the Mac OS; applications that present a graphical interface to the user should be able to any events sent by the Mac OS that apply to the application.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

Discussion

Apple events are identified by their event class and event ID attributes, each of which specifies an arbitrary four-character code. The event class appears in the `message` field of the event record for an Apple event. For example, certain Apple events that are sent by the Mac OS have the value `'aevt'` in the `message` fields of their event records. This value can be represented with the constant `kCoreEventClass`.

Groups of related Apple events are known as suites. For example, the common events that most applications support are grouped in the Standard Suite. The Standard Suite includes the events of the Core suite (`open application`, `reopen`, `open contents`, `open documents`, `print documents`, and `quit`), as well as such events as `count`, `delete`, and `make`. Suites may use a common event class, but doing so is not required, and does not result in any special treatment by AppleScript or the Apple Event Manager.

AppleScript defines suites that provide terminology for Text, Database, Macintosh Connectivity, and other types of related operations. The terms defined in the AppleScript suite itself make up the largest suite. These terms are global to AppleScript, and are available to your application, even if your `'aete'` resource doesn't explicitly include them.

Event Handler Flags

```
enum {
    kAEDoNotIgnoreHandler = 0x00000000,
    kAEIgnoreAppPhacHandler = 0x00000001,
    kAEIgnoreAppEventHandler = 0x00000002,
    kAEIgnoreSysPhacHandler = 0x00000004,
    kAEIgnoreSysEventHandler = 0x00000008,
    kAEIngoreBuiltInEventHandler = 0x00000010,
    kAEDontDisposeOnResume = 0x80000000
};
```

Event ID Constants

Specify the event ID for an Apple event.

```
enum {
    kAEOpenApplication          = 'oapp',
    kAEReopenApplication        = 'rapp',
    kAEOpenDocuments            = 'odoc',
    kAEPrintDocuments           = 'pdoc',
    kAEOpenContents             = 'ocon',
    kAEQuitApplication          = 'quit',
    kAEAnswer                   = 'ansr',
    kAEApplicationDied          = 'obit',
    kAEShowPreferences          = 'pref'
};
```

Constants

`kAEOpenApplication`

Event that launches an application.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEReopenApplication`

Event that reopens an application. Sent, for example, when your application is running and a user clicks your application icon in the Dock.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEOpenDocuments`

Event that provides an application with a list of documents to open. Sent, for example, when a selects one or more documents for your application in the Finder and double-clicks them.

See also the constant `keyAESearchText` in the enum "keyAEPropData" (page 235).

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEPrintDocuments`

Event that provides an application with a list of documents to print.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEOpenContents`

Event that provides an application with dragged content, such as text or an image. Sent, for example, when a user drags an image file onto your application's icon in the Dock. The application can use the content as desired—for example, if no document is currently open, it might open a new document and insert the provided text or image.

For more information, see "Handling Apple Events Sent by the Mac OS" in "Responding to Apple Events" in Apple Events Programming Guide.

Available in Mac OS X v10.4 and later.

Declared in `AppleEvents.h`.

`kAEQuitApplication`

Event that causes the application to quit.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEAnswer`
Event that is a reply Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEApplicationDied`
Event sent by the Process Manager to an application that launched another application when the launched application quits or terminates.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEShowPreferences`
Event sent by the Mac OS X to a process when the user chooses the Preferences item for that process.

Carbon applications that handle the Preferences command can install an Apple event handler for this event, but they more commonly install a Carbon event handler for `kEventCommandProcess` and check for the `kHICommandPreferences` command ID.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

Discussion
Apple events are identified by their event class and event ID attributes. The event ID is the attribute that identifies the particular Apple event within its 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. The event ID appears in the `where` field of the event record for an Apple event. For example, an event with ID `kAEOpenApplication` and class `kCoreEventClass` is an event sent by the Mac OS that launches an application.

Only a small number of event IDs are shown here. For a more complete listing, see the Apple Event Manager and Open Scripting Architecture header files.

Event Source Constants

Identify how an Apple event was delivered.

```
enum {
    kAEUnknownSource = 0,
    kAEDirectCall = 1,
    kAESameProcess = 2,
    kAELocalProcess = 3,
    kAERemoteProcess = 4
};
```

Constants
`kAEUnknownSource`
The source of the Apple event is unknown.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAEDirectCall`
The source of the Apple event is a direct call that bypassed the PPC Toolbox.

Available in Mac OS X v10.0 and later.

Declared in `AppleEvents.h`.

`kAESameProcess`
: The source of the Apple event is the same application that received the event (the target application and the source application are the same).

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAELocalProcess`
: The source application is another process on the same computer as the target application.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`kAERemoteProcess`
: The source application is a process on a remote computer on the network.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Discussion
For an example of how you might use these constants with the `AEGetAttributePtr` (page 46) function, see the data type `AEEventSource` (page 170).

Declared In
`AppleEvents.h`

Factoring Constants

```
enum {
    kAEDescListFactorNone = 0,
    kAEDescListFactorType = 4,
    kAEDescListFactorTypeAndSize = 8
};
```

Discussion
These constants have no effect in Mac OS X v10.2 and later.

ID Constants for the AECreateAppleEvent Function

Specify values for the ID parameters of the `AECreateAppleEvent` function.

```
enum {
    kAutoGenerateReturnID = -1,
    kAnyTransactionID = 0
};
```

Constants
`kAutoGenerateReturnID`
: If you pass this value for the `returnID` parameter of the `AECreateAppleEvent` (page 32) function, the Apple Event Manager assigns to the created Apple event a return ID that is unique to the current session.

  Available in Mac OS X v10.0 and later.

  Declared in `AEDataModel.h`.

`kAnyTransactionID`

You pass this value for the `transactionID` parameter of the `AECreateAppleEvent` (page 32) function if the Apple event is not one of a series of interdependent Apple events.

A transaction is a sequence of Apple events that are sent back and forth between the client and server applications, beginning with the client's initial request for a service. All Apple events that are part of a transaction must have the same transaction ID.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

You use these constants with the `AECreateAppleEvent` (page 32) function.

Key Form and Descriptor Type Object Specifier Constants

Specify possible values for the `keyAEKeyForm` field of an object specifier, as well as descriptor types used in resolving object specifiers.

```
enum {
    formAbsolutePosition = 'indx',
    formRelativePosition = 'rele',
    formTest = 'test',
    formRange = 'rang',
    formPropertyID = 'prop',
    formName = 'name',
    typeObjectSpecifier = 'obj ',
    typeObjectBeingExamined = 'exmn',
    typeCurrentContainer = 'ccnt',
    typeToken = 'toke',
    typeRelativeDescriptor = 'rel ',
    typeAbsoluteOrdinal = 'abso',
    typeIndexDescriptor = 'inde',
    typeRangeDescriptor = 'rang',
    typeLogicalDescriptor = 'logi',
    typeCompDescriptor = 'cmpd',
    typeOSLTokenList = 'ostl'
};
```

Constants

`formAbsolutePosition`

An integer or other constant indicating the position of one or more elements in relation to the beginning or end of their container. The key data consists of an integer that specifies either an offset or an ordinal position.

For descriptor type `typeAbsoluteOrdinal`, the data consists of one of the constants `kAEFirst`, `kAEMiddle`, `kAELast`, `kAEAny`, or `kAEAll`, which are described in `AEDisposeToken` (page 41).

For other descriptor types, the data can be coerced to either a positive integer, indicating the offset of the requested element from the beginning of the container, or a negative integer, indicating its offset from the end of the container.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formRelativePosition`

Specifies an element position either immediately before or immediately after a container, not inside it. The key data is specified by a descriptor of type `typeEnumerated` whose data consists of one of the constants `kAENext` and `kAEPrevious`, which are described in `AEDisposeToken` (page 41).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formTest`

Specifies a test. The key data is specified by either a comparison descriptor or a logical descriptor.

The Apple Event Manager internally translates object specifiers of key form `formTest` into object specifiers of key form `formWhose` to optimize resolution of object specifiers. This involves collapsing the key form and key data from two object specifiers in a container hierarchy into one object specifier with the key form `formWhose`.

See also `AEDisposeToken` (page 41), "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191), `CreateCompDescriptor` (page 114), and `CreateLogicalDescriptor` (page 115).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formRange`

Specifies a group of elements between two other elements. The key data is specified by a range descriptor, which is a coerced Apple event record of type `typeRangeDescriptor` that identifies two Apple event objects marking the beginning and end of a range of elements.

The data for a range descriptor consists of two keyword-specified descriptors with the keywords `keyAERangeStart` and `keyAERangeStop`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formPropertyID`

Specifies the property ID for an element's property.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`formName`

Specifies the Apple event object by name.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeObjectSpecifier`

Specifies a descriptor used with the `keyAEContainer` keyword in a keyword-specified descriptor. The key data for the descriptor is an object specifier.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeObjectBeingExamined`

Specifies a descriptor that acts as a placeholder for each of the successive elements in a container when the Apple Event Manager tests those elements one at a time. The descriptor has a null data storage pointer. This descriptor type is used only with `formTest`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeCurrentContainer`
Specifies a container for an element that demarcates one boundary in a range. The descriptor has a null data storage pointer. This descriptor type is used only with `formRange`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeToken`
Specifies a descriptor whose data storage pointer refers to a structure of type `AEDisposeToken` (page 41).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeRelativeDescriptor`
Specifies a descriptor whose data consists of one of the constants `kAENext` or `kAEPrevious`, which are described in `AEDisposeToken` (page 41). Used with `formRelativePosition`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeAbsoluteOrdinal`
Specifies a descriptor whose data consists of one of the constants `kAEFirst`, `kAEMiddle`, `kAELast`, `kAEAny`, or `kAEAll`, which are described in `AEDisposeToken` (page 41). Used with `formAbsolutePosition`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeIndexDescriptor`
Specifies a descriptor whose data indicates an indexed position within a range of values.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeRangeDescriptor`
Specifies a range descriptor that identifies two Apple event objects marking the beginning and end of a range of elements. The data for a range descriptor consists of two keyword-specified descriptors with the keywords `keyAERangeStart` and `keyAERangeStop`, respectively, which specify the first Apple event object in the desired range and the last Apple event object in the desired range.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeLogicalDescriptor`
Specifies a logical descriptor. Data is one of the constants described in `AEDisposeToken` (page 41).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeCompDescriptor`
Specifies a comparison descriptor. Data is one of the constants described in `AEDisposeToken` (page 41).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`typeOSLTokenList`
Specifies a descriptor whose data consists of a list of tokens. (Token is defined in `AEDisposeToken` (page 41).)

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion
The constants in this enum that begin with "form" specify the key form for an object specifier. The key form indicates how key data should be interpreted. Key form is one of the keyword-specified descriptors described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191).

The constants in this enum that begin with "type" specify descriptor types used in resolving object specifiers. An object specifier is a coerced Apple event record of descriptor type `typeObjectSpecifier` whose data consists of the four keyword-specified descriptors described in "Constants for Object Specifiers, Positions, and Logical and Comparison Operations" (page 191). One of those four keyword-specified descriptors has the type `keyAEKeyData`. This descriptor can contain data or nested descriptors specified by any of the descriptor type constants defined here (or by types defined by your application).

Keyword Attribute Constants

Specify keyword values for Apple event attributes.

```
enum {
    keyTransactionIDAttr = 'tran',
    keyReturnIDAttr = 'rtid',
    keyEventClassAttr = 'evcl',
    keyEventIDAttr = 'evid',
    keyAddressAttr = 'addr',
    keyOptionalKeywordAttr = 'optk',
    keyTimeoutAttr = 'timo',
    keyInteractLevelAttr = 'inte',
    keyEventSourceAttr = 'esrc',
    keyMissedKeywordAttr = 'miss',
    keyOriginalAddressAttr = 'from',
    keyAcceptTimeoutAttr = 'actm',
    keyReplyRequestedAttr = 'repq'
};
```

Constants
`keyTransactionIDAttr`
Transaction ID identifying a series of Apple events that are part of one transaction.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyReturnIDAttr`
Return ID for a reply Apple event.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventClassAttr`
Event class of an Apple event. See `AEAddressDesc` (page 167).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventIDAttr`
Event ID of an Apple event. See `AEAddressDesc` (page 167).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyAddressAttr`
Address of a target or client application. See also `AEAddressDesc` (page 167).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyOptionalKeywordAttr`
List of keywords for parameters of an Apple event that should be treated as optional by the target application.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyTimeoutAttr`
Length of time, in ticks, that the client will wait for a reply or a result from the server.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyInteractLevelAttr`
Settings for when to allow the Apple Event Manager to bring a server application to the foreground, if necessary, to interact with the user. See `AEAddressDesc` (page 167). (Read only.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyEventSourceAttr`
Nature of the source application. (Read only.)

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyMissedKeywordAttr`
Keyword for first required parameter remaining in an Apple event. (Read only.)

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

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyOriginalAddressAttr`
Address of original source of Apple event if the event has been forwarded (available only in version 1.01 or later versions of the Apple Event Manager). See also `AEAddressDesc` (page 167).

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`keyReplyRequestedAttr`
A Boolean value indicating whether the Apple event expects to be replied to.

Available in Mac OS X v10.3 and later.

Declared in `AEDataModel.h`.

Discussion
These constants are keyword constants for Apple event attributes. An Apple event consists of attributes (which identify the Apple event and denote its task) and, often, parameters (which contain information to be used by the target application). An Apple event attribute is a descriptor that identifies the event class, event ID, target application, or some other characteristic of the Apple event. Taken together, the attributes of an Apple event denote the task to be performed on any data specified in the Apple event's parameters.

Keywords are arbitrary names used by the Apple Event Manager to keep track of various descriptors. Your application cannot examine the contents of an Apple event directly. Instead, you call Apple Event Manager routines such as those described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 18) to request attributes and parameters by keyword.

See also "Keyword Parameter Constants" (page 211).

Version Notes
The constant `keyReplyRequestedAttr` was added in Mac OS X version 10.3.

Keyword Parameter Constants

Specify keyword values for Apple event parameters, as well as information for the `AEManagerInfo` function to retrieve. Some common key word values are shown here.

```
enum {
    keyDirectObject = '----',
    keyErrorNumber = 'errn',
    keyErrorString = 'errs',
    keyProcessSerialNumber = 'psn ',
    keyPreDispatch = 'phac',
    keySelectProc = 'selh',
    keyAERecorderCount = 'recr',
    keyAEVersion = 'vers'
};
```

Constants

`keyDirectObject`
Direct parameter. Usually specifies the data to be acted upon by the target application.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyErrorNumber`
Error number. Often used to extract error information from a reply Apple event.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyErrorString`
Error string. Often used to extract error information from a reply Apple event to display to the user.
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyProcessSerialNumber`
Process serial number. See also `AEManagerInfo` (page 70).
Available in Mac OS X v10.0 and later.
Declared in `AppleEvents.h`.

`keyPreDispatch`
: A predispatch handler (an Apple event handler that the Apple Event Manager calls immediately before it dispatches an Apple event). See also "Managing Special Handler Dispatch Tables" (page 20).

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keySelectProc`
: You pass this value in the `functionClass` parameter of the `AEManagerInfo` (page 70) function to disable the Object Support Library. Disabling the Object Support Library is not recommended.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keyAERecorderCount`
: Used with the `keyword` parameter of the `AEManagerInfo` (page 70) function. If you pass this value, on return, the `result` parameter supplies the number of processes that are currently recording Apple events.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

`keyAEVersion`
: Used with the `keyword` parameter of the `AEManagerInfo` (page 70) function. If you pass this value, on return, the `result` parameter supplies version information for the Apple Event Manager, in NumVersion format.

  Available in Mac OS X v10.0 and later.

  Declared in `AppleEvents.h`.

Discussion
These constants are keyword constants for Apple event parameters. An Apple event consists of attributes (which identify the Apple event and denote its task) and, often, parameters (which contain information to be used by the target application). Taken together, the attributes of an Apple event denote the task to be performed on any data specified in the Apple event's parameters.

Keywords are arbitrary names used by the Apple Event Manager to keep track of various descriptors. Your application cannot examine the contents of an Apple event directly. Instead, you call Apple Event Manager routines such as those described in "Getting Data or Descriptors From Apple Events and Apple Event Records" (page 18) to request attributes and parameters by keyword.

See also "Keyword Attribute Constants" (page 209).

Launch Apple Event Constants

In a `kAEOpenApplication` event, specify information about how the receiving application was launched.

```
enum {
    keyAELaunchedAsLogInItem = 'lgit',
    keyAELaunchedAsServiceItem = 'svit'
};
```

Constants

`keyAELaunchedAsLogInItem`

If present in a `kAEOpenApplication` event, the receiving application was launched as a login item and should only perform actions suitable to that environment—for example, it probably shouldn't open an untitled document.

Available in Mac OS X v10.5 and later.

Declared in `AERegistry.h`.

`keyAELaunchedAsServiceItem`

If present in a `kAEOpenApplication` event, the receiving application was launched as a service item and should only perform actions suitable to that environment—for example, it probably shouldn't open an untitled document.

Available in Mac OS X v10.5 and later.

Declared in `AERegistry.h`.

Special Considerations

Although these constants were not publicly defined in Mac OS X version 10.4, corresponding information was provided in `kAEOpenApplication` Apple events sent by that version of the OS. Therefore your application, running on Mac OS X version 10.4 or later, can examine the open application Apple event to determine if the application was launched as a login item or a service item. However, for version 10.4, you will have to define these constants in your own code file.

You check for a `keyAEPropData` parameter of the `kAEOpenApplication` Apple event, with a data value that matches `keyAELaunchedAsLogInItem` or `keyAELaunchedAsServiceItem`.

Declared In

`AERegistry.h`

Numeric Descriptor Type Constants

Specify types for numeric descriptors.

```
enum {
    typeSInt16 = 'shor',
    typeUInt16 = 'ushr',
    typeSInt32 = 'long',
    typeUInt32 = 'magn',
    typeSInt64 = 'comp',
    typeUInt64 = 'ucom',
    typeIEEE32BitFloatingPoint = 'sing',
    typeIEEE64BitFloatingPoint = 'doub',
    type128BitFloatingPoint = 'ldbl',
    typeDecimalStruct = 'decm'
};
```

Constants

`typeSInt16`
16-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt16`
16-bit unsigned integer.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeSInt32`
32-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt32`
32-bit unsigned integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeSInt64`
64-bit signed integer.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeUInt64`
64-bit unsigned integer.
Available in Mac OS X v10.5 and later.
Declared in `AEDataModel.h`.

`typeIEEE32BitFloatingPoint`
32-bit floating point value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`typeIEEE64BitFloatingPoint`
64-bit floating point value.
Available in Mac OS X v10.0 and later.
Declared in `AEDataModel.h`.

`type128BitFloatingPoint`
128-bit floating point value.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeDecimalStruct`
Decimal.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
The constants described here specify the data type for a descriptor and show the kind of numeric data stored in a descriptor with that type. These constants are preferred over their older equivalents described in "typeSMInt" (page 248).

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 162), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 176).

AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Descriptor Type Constants" (page 197) and "Other Descriptor Type Constants" (page 217). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

Declared In
`AEDataModel.h`

Object Class ID Constants

Specify the object class for an Apple event object.

```
enum {
    cParagraph = 'cpar',
    cPICT = 'PICT',
    cPixel = 'cpxl',
    cPixelMap = 'cpix',
    cPolygon = 'cpgn',
    cProperty = 'prop',
    cQDPoint = 'QDpt',
    cQDRectangle = 'qdrt',
    cRectangle = 'crec',
    cRGBColor = 'cRGB',
    cRotation = 'trot',
    cRoundedRectangle = 'crrc',
    cRow = 'crow',
    cSelection = 'csel',
    cShortInteger = 'shor',
    cTable = 'ctbl',
    cText = 'ctxt',
    cTextFlow = 'cflo',
    cTextStyles = 'tsty',
    cType = 'type'
};
```

Constants

`cParagraph`

A paragraph of text.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cPICT`

A PICT format figure.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cProperty`

A property of any object class.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`cRGBColor`

An RGB color value.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

Discussion

The object class of an Apple event object is identified by an object class ID. For example, the object class for an object specifier that specifies an RGB color value is the four-character code `'cRGB'`, which can be represented by the constant `cRGBColor`.

AppleScript defines constants for a wide variety of common object classes, though only a small number are shown here. For a more complete listing, see the Apple Event Manager and Open Scripting Architecture header files.

Other Descriptor Type Constants

Specify types for Boolean and character descriptors.

```
enum {
    typeBoolean = 'bool',
    typeChar = 'TEXT'
};
```

Constants

`typeBoolean`

Boolean value—single byte with value 0 or 1.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeChar`

Unterminated string of system script characters.

See the Version Notes section below for important information.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

The constants described here specify the data type for a descriptor and show the kind of data stored in a descriptor with that type.

Descriptors are the building blocks used by the Apple Event Manager to construct Apple event attributes and parameters. A descriptor is a data structure of type `AEDesc` (page 162), which consists of data storage and a descriptor type that identifies the type of the data. A descriptor type is defined by the data type `DescType` (page 176).

AppleScript defines descriptor type constants for a wide variety of common data types. For additional types, see "Descriptor Type Constants" (page 197) and "Numeric Descriptor Type Constants" (page 213). For a complete listing, including data types such as units of length, weight, and volume, see the Apple Event Manager and Open Scripting Architecture header files.

Version Notes

On Mac OS X `typeChar` type is deprecated in favor of `typeUTF8Text` or `typeUTF16ExternalRepresentation`. For more information, see `typeUTF16ExternalRepresentation` (page 251).

Priority Constants for the AESend Function (Deprecated in Mac OS X)

Specify a value for the `sendPriority` parameter of the `AESend` function. (**Deprecated.** Not used in Mac OS X.)

```
enum {
    kAENormalPriority = 0x00000000,
    kAEHighPriority = 0x00000001
};
```

Constants

`kAENormalPriority`

The Apple Event Manager posts the event at the end of the event queue of the server process and the server processes the Apple event as soon as it has the opportunity.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kAEHighPriority`

The Apple Event Manager posts the event at the beginning of the event queue of the server process.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

For related information, see the `AESend` (page 92) function and "AESendMode" (page 182).

Version Notes

The `sendPriority` parameter of the `AESend` function is deprecated in Mac OS X.

Remote Process Dictionary Keys

Used to extract information from dictionaries with entries that describe remote processes.

```
extern const CFStringRef kAERemoteProcessURLKey;
extern const CFStringRef kAERemoteProcessNameKey;
extern const CFStringRef kAERemoteProcessUserIDKey;
extern const CFStringRef kAERemoteProcessProcessIDKey;
```

Constants

`kAERemoteProcessURLKey`

Use this key to obtain the full URL to the remote process, as a `CFURLRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessNameKey`

Use this key to obtain the visible name of the remote process, in the localization supplied by the server, as a `CFStringRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessUserIDKey`

Use this key to obtain the user ID of the remote process, if available; if so, returned as a `CFNumberRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

`kAERemoteProcessProcessIDKey`

Use this key to obtain the process ID of the remote process, if available; if so, returned as a `CFNumberRef`.

Available in Mac OS X v10.3 and later.

Declared in `AppleEvents.h`.

Declared In

`AppleEvents.h`

Resume Event Dispatch Constants

Specify event dispatching information to the `AEResumeTheCurrentEvent` function.

```
enum {
    kAENoDispatch = 0,
    kAEUseStandardDispatch = 0xFFFFFFFF
};
```

Constants

`kAENoDispatch`

Tells the Apple Event Manager that the Apple event has been completely processed and need not be dispatched.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEUseStandardDispatch`

Tells the Apple Event Manager to dispatch the resumed event using the standard dispatching scheme it uses for other Apple events.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

Discussion

You call the `AEResumeTheCurrentEvent` (page 90) function to inform the Apple Event Manager that your application wants to resume the handling of a previously suspended Apple event or that it has completed the handling of the Apple event. You pass one of the constants described here in the `dispatcher` parameter to provide dispatching information to the Apple Event Manager. You can also pass a handler universal procedure pointer.

Special Handler Callback Constants

Specify an object callback function to install, get, or remove from the special handler dispatch table.

```
enum {
    keyAERangeStart = 'star',
    keyAERangeStop = 'stop',
    keyDisposeTokenProc = 'xtok',
    keyAECompareProc = 'cmpr',
    keyAECountProc = 'cont',
    keyAEMarkTokenProc = 'mkid',
    keyAEMarkProc = 'mark',
    keyAEAdjustMarksProc = 'adjm',
    keyAEGetErrDescProc = 'indc'
};
```

Constants

`keyAERangeStart`
Specifies the first Apple event object in a desired range.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAERangeStop`
Specifies the last Apple event object in the desired range.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyDisposeTokenProc`
Token disposal function. See `OSLDisposeTokenProcPtr` (page 155).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAECompareProc`
Object-comparison function. See `OSLCompareProcPtr` (page 152).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAECountProc`
Object-counting function. See `OSLCountProcPtr` (page 154).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEMarkTokenProc`
Mark token function. See `OSLGetMarkTokenProcPtr` (page 158).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEMarkProc`
Object-marking function. See `OSLMarkProcPtr` (page 160).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEAdjustMarksProc`
Mark-adjusting function. See `OSLAdjustMarksProcPtr` (page 151).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

`keyAEGetErrDescProc`
Get error descriptor callback function. See `OSLGetErrDescProcPtr` (page 157).

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

Discussion
You use these constants with the `AEInstallSpecialHandler` (page 68), `AEGetSpecialHandler` (page 62), or `AERemoveSpecialHandler` (page 87) functions.

Timeout Constants

Specify a timeout value.

```
enum {
    kAEDefaultTimeout = -1,
    kNoTimeOut = -2
};
```

Constants
`kAEDefaultTimeout`
The timeout value is determined by the Apple Event Manager. The default timeout value is about one minute.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`kNoTimeOut`
Your application is willing to wait indefinitely. Most commonly, you instead provide a timeout value (in ticks) that will provide a reasonable amount of time for the current operation.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion
Your application can use these constants when it calls the `AEInteractWithUser` (page 69) function, or it can supply the specific amount of time (in ticks) that your handler is willing to wait for a response from the user. You can also use the constants with the `AESend` (page 92) function.

User Interaction Level Constants

Specify to the `AESetInteractionAllowed` function the conditions under which your application is willing to interact with the user.

```
enum {
    kAEInteractWithSelf = 0,
    kAEInteractWithLocal = 1,
    kAEInteractWithAll = 2
};
```

Constants

`kAEInteractWithSelf`

Indicates that the server application may interact with the user in response to an Apple event only when the client application and server application are the same—that is, only when your application is sending the Apple event to itself.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEInteractWithLocal`

Indicates that your server application may interact with the user in response to an Apple event only if the client application is on the same computer as the server application. This is the default value if your application has not called the `AESetInteractionAllowed` (page 95) function to set the interaction level explicitly.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

`kAEInteractWithAll`

Indicates that your server application may interact with the user in response to an Apple event sent from any client application on any computer.

Available in Mac OS X v10.0 and later.

Declared in `AEInteraction.h`.

Discussion

If your application does not set the user interaction level by calling the `AESetInteractionAllowed` (page 95) function, the Apple Event Manager uses `kAEInteractWithLocal` as the default value.

Declared In

`AERegistry.h`

Whose Test Constants

```
enum {
    typeWhoseDescriptor = 'whos',
    formWhose = 'whos',
    typeWhoseRange = 'wrng',
    keyAEWhoseRangeStart = 'wstr',
    keyAEWhoseRangeStop = 'wstp',
    keyAEIndex = 'kidx',
    keyAETest = 'ktst'
};
```

Constants

`formWhose`

Specifies a container of one or more objects and a test to perform on the objects.

The key data for `formWhose` is specified by a whose descriptor, which is a coerced Apple event record of descriptor type `typeWhoseDescriptor`. The data for a whose descriptor consists of two keyword-specified descriptors with the keywords `keyAEIndex` and `keyAETest`.

See also the description for `formTest`.

Available in Mac OS X v10.0 and later.

Declared in `AEObjects.h`.

kAEDoObjectsExist

```
enum {
    kAEDoObjectsExist = 'doex',
    kAEDoScript = 'dosc',
    kAEDrag = 'drag',
    kAEDuplicateSelection = 'sdup',
    kAEEditGraphic = 'edit',
    kAEEmptyTrash = 'empt',
    kAEEnd = 'end ',
    kAEEndsWith = 'ends',
    kAEEndTransaction = 'endt',
    kAEEquals = '=   ',
    kAEExpanded = 'pexp',
    kAEFast = 'fast',
    kAEFinderEvents = 'FNDR',
    kAEFormulaProtect = 'fpro',
    kAEFullyJustified = 'full',
    kAEGetClassInfo = 'qobj',
    kAEGetData = 'getd',
    kAEGetDataSize = 'dsiz',
    kAEGetEventInfo = 'gtei',
    kAEGetInfoSelection = 'sinf'
};
```

Constants

`kAEEndsWith`

The value of `operand1` ends with the value of `operand2` (for example, the string `"operand"` ends with the string `"and"`).

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAEEquals`
: The value of `operand1` is equal to the value of `operand2`

  Available in Mac OS X v10.0 and later.

  Declared in `AERegistry.h`.

`kAEFinderEvents`
: An event that the Finder accepts.

  Available in Mac OS X v10.0 and later.

  Declared in `AERegistry.h`.

kAEDebugPOSTHeader

```
enum {
    kAEDebugPOSTHeader = 0x01,
    kAEDebugReplyHeader = 0x02,
    kAEDebugXMLRequest = 0x04,
    kAEDebugXMLResponse = 0x08,
    kAEDebugXMLDebugAll = 0xFFFFFFFF
};
```

kAEGetPrivilegeSelection

```
enum {
    kAEGetPrivilegeSelection = 'sprv',
    kAEGetSuiteInfo = 'gtsi',
    kAEGreaterThan = '>   ',
    kAEGreaterThanEquals = '>=  ',
    kAEGrow = 'grow',
    kAEHidden = 'hidn',
    kAEHiQuality = 'hiqu',
    kAEImageGraphic = 'imgr',
    kAEIsUniform = 'isun',
    kAEItalic = 'ital',
    kAELeftJustified = 'left',
    kAELessThan = '<   ',
    kAELessThanEquals = '<=  ',
    kAELowercase = 'lowc',
    kAEMakeObjectsVisible = 'mvis',
    kAEMiscStandards = 'misc',
    kAEModifiable = 'modf',
    kAEMove = 'move',
    kAENo = 'no  ',
    kAENoArrow = 'arno'
};
```

Constants

`kAEGreaterThan`
: The value of `operand1` is greater than the value of `operand2`.

  Available in Mac OS X v10.0 and later.

  Declared in `AERegistry.h`.

`kAEGreaterThanEquals`
The value of `operand1` is greater than or equal to the value of `operand2`.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

`kAELessThanEquals`
The value of `operand1` is less than or equal to the value of `operand2`.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

kAEHandleArray

```
enum {
    kAEHandleArray = 2
};
```

Constants

`kAEHandleArray`
Array items consist of handles to data of the same type and possibly variable size.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

kAEInfo

```
enum {
    kAEInfo = 11,
    kAEMain = 0,
    kAESharing = 13
};
```

kAEInternetSuite

```
enum {
    kAEInternetSuite = 'gurl',
    kAEISWebStarSuite = 'WWW'
};
```

kAEISGetURL

```
enum {
    kAEISGetURL = 'gurl',
    KAEISHandleCGI = 'sdoc'
};
```

kAEISHTTPSearchArgs

```
enum {
    kAEISHTTPSearchArgs = 'kfor',
    kAEISPostArgs = 'post',
    kAEISMethod = 'meth',
    kAEISClientAddress = 'addr',
    kAEISUserName = 'user',
    kAEISPassword = 'pass',
    kAEISFromUser = 'frmu',
    kAEISServerName = 'svnm',
    kAEISServerPort = 'svpt',
    kAEISScriptName = 'scnm',
    kAEISContentType = 'ctyp',
    kAEISReferrer = 'refr',
    kAEISUserAgent = 'Agnt',
    kAEISAction = 'Kact',
    kAEISActionPath = 'Kapt',
    kAEISClientIP = 'Kcip',
    kAEISFullRequest = 'Kfrq'
};
```

kAELogOut

```
enum {
    kAELogOut = 'logo',
    kAEReallyLogOut = 'rlgo',
    kAEShowRestartDialog = 'rrst',
    kAEShowShutdownDialog = 'rsdn'
```

```
};
```

kAEMenuClass

```
enum {
    kAEMenuClass = 'menu',
    kAEMenuSelect = 'mhit',
    kAEMouseDown = 'mdwn',
    kAEMouseDownInBack = 'mdbk',
    kAEKeyDown = 'kdwn',
    kAEResized = 'rsiz',
    kAEPromise = 'prom'
};
```

kAEMouseClass

```
enum {
    kAEMouseClass = 'mous',
    kAEDown = 'down',
    kAEUp = 'up  ',
    kAEMoved = 'move',
    kAEStoppedMoving = 'stop',
    kAEWindowClass = 'wind',
    kAEUpdate = 'updt',
    kAEActivate = 'actv',
    kAEDeactivate = 'dact',
    kAECommandClass = 'cmnd',
    kAEKeyClass = 'keyc',
    kAERawKey = 'rkey',
    kAEVirtualKey = 'keyc',
    kAENavigationKey = 'nave',
    kAEAutoDown = 'auto',
    kAEApplicationClass = 'appl',
    kAESuspend = 'susp',
    kAEResume = 'rsme',
    kAEDiskEvent = 'disk',
    kAENullEvent = 'null',
    kAEWakeUpEvent = 'wake',
    kAEScrapEvent = 'scrp',
    kAEHighLevel = 'high'
};
```

kAENonmodifiable

```
enum {
    kAENonmodifiable = 'nmod',
    kAEOpen = 'odoc',
    kAEOpenSelection = 'sope',
    kAEOutline = 'outl',
    kAEPageSetup = 'pgsu',
    kAEPaste = 'past',
    kAEPlain = 'plan',
    kAEPrint = 'pdoc',
```

```
    kAEPrintSelection = 'spri',
    kAEPrintWindow = 'pwin',
    kAEPutAwaySelection = 'sput',
    kAEQDAddOver = 'addo',
    kAEQDAddPin = 'addp',
    kAEQDAdMax = 'admx',
    kAEQDAdMin = 'admn',
    kAEQDBic = 'bic ',
    kAEQDBlend = 'blnd',
    kAEQDCopy = 'cpy ',
    kAEQDNotBic = 'nbic',
    kAEQDNotCopy = 'ncpy'
};
```

kAEQDNotOr

```
enum {
    kAEQDNotOr = 'ntor',
    kAEQDNotXor = 'nxor',
    kAEQDOr = 'or  ',
    kAEQDSubOver = 'subo',
    kAEQDSubPin = 'subp',
    kAEQDSupplementalSuite = 'qdsp',
    kAEQDXor = 'xor ',
    kAEQuickdrawSuite = 'qdrw',
    kAEQuitAll = 'quia',
    kAERedo = 'redo',
    kAERegular = 'regl',
    kAEReplace = 'rplc',
    kAERequiredSuite = 'reqd',
    kAERestart = 'rest',
    kAERevealSelection = 'srev',
    kAERevert = 'rvrt',
    kAERightJustified = 'rght',
    kAESave = 'save',
    kAESelect = 'slct',
    kAESetData = 'setd'
};
```

kAESetPosition

```
enum {
    kAESetPosition = 'posn',
    kAEShadow = 'shad',
    kAEShowClipboard = 'shcl',
    kAEShutDown = 'shut',
    kAESleep = 'slep',
    kAESmallCaps = 'smcp',
    kAESpecialClassProperties = 'c@#!',
    kAEStrikethrough = 'strk',
    kAESubscript = 'sbsc',
    kAESuperscript = 'spsc',
    kAETableSuite = 'tbls',
    kAETextSuite = 'TEXT',
    kAETransactionTerminated = 'ttrm',
```

```
    kAEUnderline = 'undl',
    kAEUndo = 'undo',
    kAEWholeWordEquals = 'wweq',
    kAEYes = 'yes ',
    kAEZoom = 'zoom'
};
```

kAESocks4Protocol

```
enum {
    kAESocks4Protocol = 4,
    kAESocks5Protocol = 5
};
```

kAEUseHTTPProxyAttr

Web Services Proxy support—these constants should be added as attributes of the event that is being sent (not as part of the direct object).

```
enum {
    kAEUseHTTPProxyAttr = 'xupr',
    kAEHTTPProxyPortAttr = 'xhtp',
    kAEHTTPProxyHostAttr = 'xhth'
};
```

Constants

`kAEUseHTTPProxyAttr`

A value of type `typeBoolean`. Specifies whether to manually specify the proxy host and port. Defaults to `true`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`kAEHTTPProxyPortAttr`

A value of type `typeSInt32`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`kAEHTTPProxyHostAttr`

A value of type `typeChar` or `typeUTF8Text`.

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

kAEUserTerminology

```
enum {
    kAEUserTerminology = 'aeut',
    kAETerminologyExtension = 'aete',
    kAEScriptingSizeResource = 'scsz',
    kAEOSAXSizeResource = 'osiz'
};
```

kAEUseSocksAttr

```
enum {
    kAEUseSocksAttr = 'xscs',
    kAESocksProxyAttr = 'xsok',
    kAESocksHostAttr = 'xshs',
    kAESocksPortAttr = 'xshp',
    kAESocksUserAttr = 'xshu',
    kAESocksPasswordAttr = 'xshw'
};
```

kAEUTHasReturningParam

```
enum {
    kAEUTHasReturningParam = 31,
    kAEUTOptional = 15,
    kAEUTlistOfItems = 14,
    kAEUTEnumerated = 13,
    kAEUTReadWrite = 12,
    kAEUTChangesState = 12,
    kAEUTTightBindingFunction = 12,
    kAEUTEnumsAreTypes = 11,
    kAEUTEnumListIsExclusive = 10,
    kAEUTReplyIsReference = 9,
    kAEUTDirectParamIsReference = 9,
    kAEUTParamIsReference = 9,
    kAEUTPropertyIsReference = 9,
    kAEUTNotDirectParamIsTarget = 8,
    kAEUTParamIsTarget = 8,
    kAEUTApostrophe = 3,
    kAEUTFeminine = 2,
    kAEUTMasculine = 1,
    kAEUTPlural = 0
};
```

kAEZoomIn

```
enum {
    kAEZoomIn = 7,
    kAEZoomOut = 8
};
```

kBySmallIcon

```
enum {
    kBySmallIcon = 0,
    kByIconView = 1,
    kByNameView = 2,
    kByDateView = 3,
    kBySizeView = 4,
    kByKindView = 5,
    kByCommentView = 6,
    kByLabelView = 7,
    kByVersionView = 8
};
```

kCaretPosition

```
enum {
    kCaretPosition = 1,
    kRawText = 2,
    kSelectedRawText = 3,
    kConvertedText = 4,
    kSelectedConvertedText = 5,
    kBlockFillText = 6,
    kOutlineText = 7,
    kSelectedText = 8
};
```

Version Notes
Starting in Mac OS X v10.4, use the constants defined in "kTSMHiliteCaretPosition" (page 240) in place of these constants.

kConnSuite

```
enum {
    kConnSuite = 'macc',
    cDevSpec = 'cdev',
    cAddressSpec = 'cadr',
    cADBAddress = 'cadb',
    cAppleTalkAddress = 'cat ',
    cBusAddress = 'cbus',
    cEthernetAddress = 'cen ',
    cFireWireAddress = 'cfw ',
    cIPAddress = 'cip ',
    cLocalTalkAddress = 'clt ',
    cSCSIAddress = 'cscs',
    cTokenRingAddress = 'ctok',
    cUSBAddress = 'cusb',
    pDeviceType = 'pdvt',
    pDeviceAddress = 'pdva',
    pConduit = 'pcon',
    pProtocol = 'pprt',
    pATMachine = 'patm',
    pATZone = 'patz',
    pATType = 'patt',
    pDottedDecimal = 'pipd',
    pDNS = 'pdns',
    pPort = 'ppor',
    pNetwork = 'pnet',
    pNode = 'pnod',
    pSocket = 'psoc',
    pSCSIBus = 'pscb',
    pSCSILUN = 'pslu',
    eDeviceType = 'edvt',
    eAddressSpec = 'eads',
    eConduit = 'econ',
    eProtocol = 'epro',
    eADB = 'eadb',
    eAnalogAudio = 'epau',
    eAppleTalk = 'epat',
    eAudioLineIn = 'ecai',
    eAudioLineOut = 'ecal',
    eAudioOut = 'ecao',
    eBus = 'ebus',
    eCDROM = 'ecd ',
    eCommSlot = 'eccm',
    eDigitalAudio = 'epda',
    eDisplay = 'edds',
    eDVD = 'edvd',
    eEthernet = 'ecen',
    eFireWire = 'ecfw',
    eFloppy = 'efd ',
    eHD = 'ehd ',
    eInfrared = 'ecir',
    eIP = 'epip',
    eIrDA = 'epir',
    eIRTalk = 'epit',
    eKeyboard = 'ekbd',
    eLCD = 'edlc',
    eLocalTalk = 'eclt',
```

```
    eMacIP = 'epmi',
    eMacVideo = 'epmv',
    eMicrophone = 'ecmi',
    eModemPort = 'ecmp',
    eModemPrinterPort = 'empp',
    eModem = 'edmm',
    eMonitorOut = 'ecmn',
    eMouse = 'emou',
    eNuBusCard = 'ednb',
    eNuBus = 'enub',
    ePCcard = 'ecpc',
    ePCIbus = 'ecpi',
    ePCIcard = 'edpi',
    ePDSslot = 'ecpd',
    ePDScard = 'epds',
    ePointingDevice = 'edpd',
    ePostScript = 'epps',
    ePPP = 'eppp',
    ePrinterPort = 'ecpp',
    ePrinter = 'edpr',
    eSvideo = 'epsv',
    eSCSI = 'ecsc',
    eSerial = 'epsr',
    eSpeakers = 'edsp',
    eStorageDevice = 'edst',
    eSVGA = 'epsg',
    eTokenRing = 'etok',
    eTrackball = 'etrk',
    eTrackpad = 'edtp',
    eUSB = 'ecus',
    eVideoIn = 'ecvi',
    eVideoMonitor = 'edvm',
    eVideoOut = 'ecvo'
};
```

keyAEAngle

```
enum {
    keyAEAngle = 'kang',
    keyAEArcAngle = 'parc'
};
```

keyAEBaseAddr

```
enum {
    keyAEBaseAddr = 'badd',
    keyAEBestType = 'pbst',
    keyAEBgndColor = 'kbcl',
    keyAEBgndPattern = 'kbpt',
    keyAEBounds = 'pbnd',
    keyAECellList = 'kclt',
    keyAEClassID = 'clID',
    keyAEColor = 'colr',
    keyAEColorTable = 'cltb',
    keyAECurveHeight = 'kchd',
```

```
    keyAECurveWidth = 'kcwd',
    keyAEDashStyle = 'pdst',
    keyAEData = 'data',
    keyAEDefaultType = 'deft',
    keyAEDefinitionRect = 'pdrt',
    keyAEDescType = 'dstp',
    keyAEDestination = 'dest',
    keyAEDoAntiAlias = 'anta',
    keyAEDoDithered = 'gdit',
    keyAEDoRotate = 'kdrt'
};
```

keyAEDoScale

```
enum {
    keyAEDoScale = 'ksca',
    keyAEDoTranslate = 'ktra',
    keyAEEditionFileLoc = 'eloc',
    keyAEElements = 'elms',
    keyAEEndPoint = 'pend',
    keyAEEventClass = 'evcl',
    keyAEEventID = 'evti',
    keyAEFile = 'kfil',
    keyAEFileType = 'fltp',
    keyAEFillColor = 'flcl',
    keyAEFillPattern = 'flpt',
    keyAEFlipHorizontal = 'kfho',
    keyAEFlipVertical = 'kfvt',
    keyAEFont = 'font',
    keyAEFormula = 'pfor',
    keyAEGraphicObjects = 'gobs',
    keyAEID = 'ID  ',
    keyAEImageQuality = 'gqua',
    keyAEInsertHere = 'insh',
    keyAEKeyForms = 'keyf'
};
```

keyAEHiliteRange

```
enum {
    keyAEHiliteRange = 'hrng',
    keyAEPinRange = 'pnrg',
    keyAEClauseOffsets = 'clau',
    keyAEOffset = 'ofst',
    keyAEPoint = 'gpos',
    keyAELeftSide = 'klef',
    keyAERegionClass = 'rgnc',
    keyAEDragging = 'bool'
};
```

keyAEKeyword

```
enum {
```

```
    keyAEKeyword = 'kywd',
    keyAELevel = 'levl',
    keyAELineArrow = 'arro',
    keyAEName = 'pnam',
    keyAENewElementLoc = 'pnel',
    keyAEObject = 'kobj',
    keyAEObjectClass = 'kocl',
    keyAEOffStyles = 'ofst',
    keyAEOnStyles = 'onst',
    keyAEParameters = 'prms',
    keyAEParamFlags = 'pmfg',
    keyAEPenColor = 'ppcl',
    keyAEPenPattern = 'pppa',
    keyAEPenWidth = 'ppwd',
    keyAEPixelDepth = 'pdpt',
    keyAEPixMapMinus = 'kpmm',
    keyAEPMTable = 'kpmt',
    keyAEPointList = 'ptlt',
    keyAEPointSize = 'ptsz',
    keyAEPosition = 'kpos'
};
```

keyAELeadingEdge

```
enum {
    keyAELeadingEdge = 'klef'
};
```

keyAEPropData

```
enum {
    keyAEPropData = 'prdt',
    keyAEProperties = 'qpro',
    keyAEProperty = 'kprp',
    keyAEPropFlags = 'prfg',
    keyAEPropID = 'prop',
    keyAEProtection = 'ppro',
    keyAERenderAs = 'kren',
    keyAERequestedType = 'rtyp',
    keyAEResult = '----',
    keyAEResultInfo = 'rsin',
    keyAERotation = 'prot',
    keyAERotPoint = 'krtp',
    keyAERowList = 'krls',
    keyAESaveOptions = 'savo',
    keyAEScale = 'pscl',
    keyAEScriptTag = 'psct',
    keyAESearchText = 'stxt',
    keyAEShowWhere = 'show',
    keyAEStartAngle = 'pang',
    keyAEStartPoint = 'pstp',
    keyAEStyles = 'ksty'
};
```

Constants

`keyAESearchText`

Identifies an optional parameter to the `open documents` Apple event, described in "Event ID Constants" (page 202). The parameter contains the search text from the Spotlight search that identified the documents to be opened. The application should make a reasonable effort to display an occurrence of the search text in each opened document—for example by scrolling the text into view.

For more information, see "Handling Apple Events Sent by the Mac OS" in "Responding to Apple Events" in Apple Events Programming Guide.

Available in Mac OS X v10.4 and later.

Declared in `AERegistry.h`.

Version Notes

The constant `keyAESearchText` is available starting in Mac OS X v10.4.

keyAESuiteID

```
enum {
    keyAESuiteID = 'suit',
    keyAEText = 'ktxt',
    keyAETextColor = 'ptxc',
    keyAETextFont = 'ptxf',
    keyAETextPointSize = 'ptps',
    keyAETextStyles = 'txst',
    keyAETextLineHeight = 'ktlh',
    keyAETextLineAscent = 'ktas',
    keyAETheText = 'thtx',
    keyAETransferMode = 'pptm',
    keyAETranslation = 'ptrs',
    keyAETryAsStructGraf = 'toog',
    keyAEUniformStyles = 'ustl',
    keyAEUpdateOn = 'pupd',
    keyAEUserTerm = 'utrm',
    keyAEWindow = 'wndw',
    keyAEWritingCode = 'wrcd'
};
```

keyMenuID

```
enum {
    keyMenuID = 'mid ',
    keyMenuItem = 'mitm',
    keyCloseAllWindows = 'caw ',
    keyOriginalBounds = 'obnd',
    keyNewBounds = 'nbnd',
    keyLocalWhere = 'lwhr'
};
```

keyMiscellaneous

```
enum {
    keyMiscellaneous = 'fmsc',
    keySelection = 'fsel',
    keyWindow = 'kwnd',
    keyWhen = 'when',
    keyWhere = 'wher',
    keyModifiers = 'mods',
    keyKey = 'key ',
    keyKeyCode = 'code',
    keyKeyboard = 'keyb',
    keyDriveNumber = 'drv#',
    keyErrorCode = 'err#',
    keyHighLevelClass = 'hcls',
    keyHighLevelID = 'hid '
};
```

keyReplyPortAttr

```
enum {
    keyReplyPortAttr = 'repp'
};
```

keySOAPStructureMetaData

```
enum {
    keySOAPStructureMetaData = '/smd',
    keySOAPSMDNamespace = 'ssns',
    keySOAPSMDNamespaceURI = 'ssnu',
    keySOAPSMDType = 'sstp'
};
```

keyUserNameAttr

```
enum {
    keyUserNameAttr = 'unam',
    keyUserPasswordAttr = 'pass',
    keyDisableAuthenticationAttr = 'auth',
    keyXMLDebuggingAttr = 'xdbg',
    kAERPCClass = 'rpc ',
    kAEXMLRPCScheme = 'RPC2',
    kAESOAPScheme = 'SOAP',
    kAESharedScriptHandler = 'wscp',
    keyRPCMethodName = 'meth',
    keyRPCMethodParam = 'parm',
    keyRPCMethodParamOrder = '/ord',
    keyAEPOSTHeaderData = 'phed',
    keyAEReplyHeaderData = 'rhed',
    keyAEXMLRequestData = 'xreq',
    keyAEXMLReplyData = 'xrep',
    keyAdditionalHTTPHeaders = 'ahed',
    keySOAPAction = 'sact',
    keySOAPMethodNameSpace = 'mspc',
    keySOAPMethodNameSpaceURI = 'mspu',
    keySOAPSchemaVersion = 'ssch'
};
```

kFAServerApp

```
enum {
    kFAServerApp = 'ssrv',
    kDoFolderActionEvent = 'fola',
    kFolderActionCode = 'actn',
    kFolderOpenedEvent = 'fopn',
    kFolderClosedEvent = 'fclo',
    kFolderWindowMovedEvent = 'fsiz',
    kFolderItemsAddedEvent = 'fget',
    kFolderItemsRemovedEvent = 'flos',
    kItemList = 'flst',
    kNewSizeParameter = 'fnsz',
    kFASuiteCode = 'faco',
    kFAAttachCommand = 'atfa',
```

```
    kFARemoveCommand = 'rmfa',
    kFAEditCommand = 'edfa',
    kFAFileParam = 'faal',
    kFAIndexParam = 'indx'
};
```

kLaunchToGetTerminology

```
enum {
    kLaunchToGetTerminology = 0x8000,
    kDontFindAppBySignature = 0x4000,
    kAlwaysSendSubject = 0x2000
};
```

kNextBody

```
enum {
    kNextBody = 1,
    kPreviousBody = 2
};
```

kOSIZDontOpenResourceFile

```
enum {
    kOSIZDontOpenResourceFile = 15,
    kOSIZdontAcceptRemoteEvents = 14,
    kOSIZOpenWithReadPermission = 13,
    kOSIZCodeInSharedLibraries = 11
};
```

kReadExtensionTermsMask

```
enum {
    kReadExtensionTermsMask = 0x8000
};
```

kSOAP1999Schema

```
enum {
    kSOAP1999Schema = 'ss99',
    kSOAP2001Schema = 'ss01'
};
```

kTextServiceClass

```
enum {
    kTextServiceClass = 'tsvc',
    kUpdateActiveInputArea = 'updt',
```

```
    kShowHideInputWindow = 'shiw',
    kPos2Offset = 'p2st',
    kOffset2Pos = 'st2p',
    kUnicodeNotFromInputMethod = 'unim',
    kGetSelectedText = 'gtxt',
    keyAETSMDocumentRefcon = 'refc',
    keyAEServerInstance = 'srvi',
    keyAETheData = 'kdat',
    keyAEFixLength = 'fixl',
    keyAEUpdateRange = 'udng',
    keyAECurrentPoint = 'cpos',
    keyAEBufferSize = 'buff',
    keyAEMoveView = 'mvvw',
    keyAENextBody = 'nxbd',
    keyAETSMScriptTag = 'sclg',
    keyAETSMTextFont = 'ktxf',
    keyAETSMTextFMFont = 'ktxm',
    keyAETSMTextPointSize = 'ktps',
    keyAETSMEventRecord = 'tevt',
    keyAETSMEventRef = 'tevr',
    keyAETextServiceEncoding = 'tsen',
    keyAETextServiceMacEncoding = 'tmen',
    keyAETSMGlyphInfoArray = 'tgia',
    typeTextRange = 'txrn',
    typeComponentInstance = 'cmpi',
    typeOffsetArray = 'ofay',
    typeTextRangeArray = 'tray',
    typeLowLevelEventRecord = 'evtr',
    typeGlyphInfoArray = 'glia',
    typeEventRef = 'evrf',
    typeText = 'TEXT'
};
```

kTSMHiliteCaretPosition

Specify text highlighting information.

```
enum {
    kTSMHiliteCaretPosition = 1,
    kTSMHiliteRawText = 2,
    kTSMHiliteSelectedRawText = 3,
    kTSMHiliteConvertedText = 4,
    kTSMHiliteSelectedConvertedText = 5,
    kTSMHiliteBlockFillText = 6,
    kTSMHiliteOutlineText = 7,
    kTSMHiliteSelectedText = 8,
    kTSMHiliteNoHilite = 9
};
```

Constants

`kTSMHiliteCaretPosition`

Specifies caret position.

Available in Mac OS X v10.4 and later.

Declared in `AERegistry.h`.

`kTSMHiliteRawText`
Specifies range of raw text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedRawText`
Specifies range of selected raw text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteConvertedText`
Specifies range of converted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedConvertedText`
Specifies range of selected converted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteBlockFillText`
Specifies block fill highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteOutlineText`
Specifies outline highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteSelectedText`
Specifies selected highlight style.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

`kTSMHiliteNoHilite`
Specifies range of non-highlighted text.
Available in Mac OS X v10.4 and later.
Declared in `AERegistry.h`.

Version Notes
This enumeration is available starting in Mac OS X v10.4. Use these constants in place of the constants defined in "kCaretPosition" (page 231).

kTSMOutsideOfBody

```
enum {
    kTSMOutsideOfBody = 1,
    kTSMInsideOfBody = 2,
    kTSMInsideOfActiveInputArea = 3
};
```

pArcAngle

```
enum {
    pArcAngle = 'parc',
    pBackgroundColor = 'pbcl',
    pBackgroundPattern = 'pbpt',
    pBestType = 'pbst',
    pBounds = 'pbnd',
    pClass = 'pcls',
    pClipboard = 'pcli',
    pColor = 'colr',
    pColorTable = 'cltb',
    pContents = 'pcnt',
    pCornerCurveHeight = 'pchd',
    pCornerCurveWidth = 'pcwd',
    pDashStyle = 'pdst',
    pDefaultType = 'deft',
    pDefinitionRect = 'pdrt',
    pEnabled = 'enbl',
    pEndPoint = 'pend',
    pFillColor = 'flcl',
    pFillPattern = 'flpt',
    pFont = 'font'
};
```

pFormula

```
enum {
    pFormula = 'pfor',
    pGraphicObjects = 'gobs',
    pHasCloseBox = 'hclb',
    pHasTitleBar = 'ptit',
    pID = 'ID  ',
    pIndex = 'pidx',
    pInsertionLoc = 'pins',
    pIsFloating = 'isfl',
    pIsFrontProcess = 'pisf',
    pIsModal = 'pmod',
    pIsModified = 'imod',
    pIsResizable = 'prsz',
    pIsStationeryPad = 'pspd',
    pIsZoomable = 'iszm',
    pIsZoomed = 'pzum',
    pItemNumber = 'itmn',
    pJustification = 'pjst',
    pLineArrow = 'arro',
    pMenuID = 'mnid',
```

```
    pName = 'pnam'
};
```

pNewElementLoc

```
enum {
    pNewElementLoc = 'pnel',
    pPenColor = 'ppcl',
    pPenPattern = 'pppa',
    pPenWidth = 'ppwd',
    pPixelDepth = 'pdpt',
    pPointList = 'ptlt',
    pPointSize = 'ptsz',
    pProtection = 'ppro',
    pRotation = 'prot',
    pScale = 'pscl',
    pScript = 'scpt',
    pScriptTag = 'psct',
    pSelected = 'selc',
    pSelection = 'sele',
    pStartAngle = 'pang',
    pStartPoint = 'pstp',
    pTextColor = 'ptxc',
    pTextFont = 'ptxf',
    pTextItemDelimiters = 'txdl',
    pTextPointSize = 'ptps'
};
```

pScheme

```
enum {
    pScheme = 'pusc',
    pHost = 'HOST',
    pPath = 'FTPc',
    pUserName = 'RAun',
    pUserPassword = 'RApw',
    pDNSForm = 'pDNS',
    pURL = 'pURL',
    pTextEncoding = 'ptxe',
    pFTPKind = 'kind'
};
```

pTextStyles

```
enum {
    pTextStyles = 'txst',
    pTransferMode = 'pptm',
    pTranslation = 'ptrs',
    pUniformStyles = 'ustl',
    pUpdateOn = 'pupd',
    pUserSelection = 'pusl',
    pVersion = 'vers',
    pVisible = 'pvis'
```

```
};
```

typeAEText

```
enum {
    typeAEText = 'tTXT',
    typeArc = 'carc',
    typeBest = 'best',
    typeCell = 'ccel',
    typeClassInfo = 'gcli',
    typeColorTable = 'clrt',
    typeColumn = 'ccol',
    typeDashStyle = 'tdas',
    typeData = 'tdta',
    typeDrawingArea = 'cdrw',
    typeElemInfo = 'elin',
    typeEnumeration = 'enum',
    typeEPS = 'EPS ',
    typeEventInfo = 'evin'
};
```

typeApplicationBundleID

For specifying a target application by bundle ID.

```
enum {
    typeApplicationBundleID = 'bund'
};
```

Constants

`typeApplicationBundleID`

Indicates a descriptor containing UTF-8 characters that specify the bundle ID of an application. Bundle IDs should be constructed similarly to `"com.company.directorylocation.ApplicationName"`.

Available in Mac OS X v10.3 and later.

Declared in `AEDataModel.h`.

Discussion

This address mode is preferred for targeting specific applications. For example, you should target the Finder by sending an event whose target address descriptor uses the bundle ID `"com.apple.finder"` rather than the application signature `'MACS'`.

typeFinderWindow

```
enum {
    typeFinderWindow = 'fwin',
    typeFixedPoint = 'fpnt',
    typeFixedRectangle = 'frct',
    typeGraphicLine = 'glin',
    typeGraphicText = 'cgtx',
    typeGroupedGraphic = 'cpic',
    typeInsertionLoc = 'insl',
    typeIntlText = 'itxt',
    typeIntlWritingCode = 'intl',
    typeLongDateTime = 'ldt ',
    typeISO8601DateTime = 'isot',
    typeLongFixed = 'lfxd',
    typeLongFixedPoint = 'lfpt',
    typeLongFixedRectangle = 'lfrc',
    typeLongPoint = 'lpnt',
    typeLongRectangle = 'lrct',
    typeMachineLoc = 'mLoc',
    typeOval = 'covl',
    typeParamInfo = 'pmin',
    typePict = 'PICT'
};
```

Constants

`typeIntlText`

For important information, see the Version Notes section of the “`typeUnicodeText`” (page 251) enum.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

typeHIMenu

```
enum {
    typeHIMenu = 'mobj',
    typeHIWindow = 'wobj'
};
```

typeKernelProcessID

For specifying an application by UNIX process ID.

```
enum {
    typeKernelProcessID = 'kpid'
};
```

Constants

`typeKernelProcessID`

Indicates a descriptor containing a UNIX process ID. A process ID is similar to a PSN (processor serial number) but does not require a Process Manager connection. It is analogous to a 32-bit unsigned integer.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

You might use this constant in a situation where you have access to the PID for a process but don't have a Process Manager connection. Your code for creating the descriptor might look like the following:

```
pid_t pid = findTheAppPid(); // User supplied routine to get PID. // Now create
 a descriptor with it: AECreateDesc(typeKernelProcessID, &pid, sizeof(pid),
&desc);
```

typeMachPort

For specifying a Mach port.

```
enum {
    typeMachPort = 'port'
};
```

Constants

`typeMachPort`

Indicates a descriptor that specifies a Mach port.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Discussion

You might use this constant as part of sending an Apple event to an arbitrary Mach port. Your code for creating the descriptor might look like the following:

```
mach_port_t port = lookupPortForTarget(); // User routine to get  port.
// Now create a descriptor with it:
AECreateDesc(typeMachPort, &port, sizeof(port), &desc);
```

Actually sending an Apple event to a Mach port is an advanced technique and is not documented here.

typeMeters

```
enum {
    typeMeters = 'metr',
    typeInches = 'inch',
    typeFeet = 'feet',
    typeYards = 'yard',
    typeMiles = 'mile',
    typeKilometers = 'kmtr',
    typeCentimeters = 'cmtr',
    typeSquareMeters = 'sqrm',
    typeSquareFeet = 'sqft',
    typeSquareYards = 'sqyd',
    typeSquareMiles = 'sqmi',
    typeSquareKilometers = 'sqkm',
    typeLiters = 'litr',
    typeQuarts = 'qrts',
    typeGallons = 'galn',
    typeCubicMeters = 'cmet',
    typeCubicFeet = 'cfet',
    typeCubicInches = 'cuin',
    typeCubicCentimeter = 'ccmt',
    typeCubicYards = 'cyrd',
    typeKilograms = 'kgrm',
    typeGrams = 'gram',
    typeOunces = 'ozs ',
    typePounds = 'lbs ',
    typeDegreesC = 'degc',
    typeDegreesF = 'degf',
    typeDegreesK = 'degk'
};
```

typePixelMap

```
enum {
    typePixelMap = 'cpix',
    typePixMapMinus = 'tpmm',
    typePolygon = 'cpgn',
    typePropInfo = 'pinf',
    typePtr = 'ptr ',
    typeQDPoint = 'QDpt',
    typeQDRegion = 'Qrgn',
    typeRectangle = 'crec',
    typeRGB16 = 'tr16',
    typeRGB96 = 'tr96',
    typeRGBColor = 'cRGB',
    typeRotation = 'trot',
    typeRoundedRectangle = 'crrc',
    typeRow = 'crow',
    typeScrapStyles = 'styl',
    typeScript = 'scpt',
    typeStyledText = 'STXT',
    typeSuiteInfo = 'suin',
    typeTable = 'ctbl',
    typeTextStyles = 'tsty'
};
```

Constants

`typeStyledText`

Text that includes style information.

Styled text is stored as a record, in which the styles have the key `'ksty'` and the plain text is has the key `'ktxt'`. You can use this information to extract plain text from styled text without coercion.

However, getting rid of the style information, with or without coercion, may corrupt the text, since the styles imply what encoding to use. In fact, use of `typeText` and `typeStyledText` are not recommended, starting with Mac OS X, because they are not safe with international characters—you should use one of the Unicode text types instead.

For important information, see the Version Notes section of the "`typeUnicodeText`" (page 251) enum.

Available in Mac OS X v10.0 and later.

Declared in `AERegistry.h`.

typeReplyPortAttr

```
enum {
    typeReplyPortAttr = 'repp'
};
```

typeSessionID

```
enum {
    typeSessionID = 'ssid',
    typeTargetID = 'targ',
    typeDispatcherID = 'dspt'
};
```

Constants

`typeSessionID`

Session reference number.

`typeTargetID`

Target ID descriptor. Target IDs are not supported in Mac OS X.

typeSMInt

Where possible, you should use the constants defined in "Numeric Descriptor Type Constants" (page 213), rather than those defined here.

```
enum {
    typeSMInt = 'shor',
    typeShortInteger = 'shor',
    typeInteger = 'long',
    typeLongInteger = 'long',
    typeMagnitude = 'magn',
    typeComp = 'comp',
    typeSMFloat = 'sing',
    typeShortFloat = 'sing',
    typeFloat = 'doub',
    typeLongFloat = 'doub',
    typeExtended = 'exte'
};
```

Constants

`typeSMInt`

16-bit integer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeShortInteger`

16-bit integer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeInteger`

32-bit integer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeLongInteger`

32-bit integer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeMagnitude`

Unsigned 32-bit integer.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeComp`

Standard Apple Numerics Environment (SANE) comparison operator.

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared in `AEDataModel.h`.

`typeSMFloat`
SANE single.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeShortFloat`
SANE single.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeFloat`
SANE double.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeLongFloat`
SANE double.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

`typeExtended`
SANE extended.
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.
Declared in `AEDataModel.h`.

typeTIFF

```
enum {
    typeTIFF = 'TIFF',
    typeVersion = 'vers'
};
```

typeUnicodeText

```
enum {
    typeUTF16ExternalRepresentation = 'ut16',
    typeUnicodeText = 'utxt',
    typeStyledUnicodeText = 'sutx',
    typeUTF8Text = 'utf8',
    typeEncodedString = 'encs',
    typeCString = 'cstr',
    typePString = 'pstr'
};
```

Constants

`typeUTF16ExternalRepresentation`
: Unicode text in 16-bit external representation with byte-order-mark (BOM).

Guarantees that either there is a BOM or the data is in UTF-16BE.

Available in Mac OS X v10.4 and later.

Declared in `AEDataModel.h`.

`typeUnicodeText`
: Unicode text. Native byte ordering, optional BOM.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeStyledUnicodeText`
: Styled Unicode text. Not implemented.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeUTF8Text`
: 8-bit Unicode (UTF-8 encoding).

Available in Mac OS X v10.2 and later.

Declared in `AEDataModel.h`.

`typeEncodedString`
: Styled Unicode text. Not implemented.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typeCString`
: C string—Mac OS Roman characters followed by a NULL byte. Deprecated.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

`typePString`

Pascal string—unsigned length byte followed by Mac OS Roman characters. Deprecated.

Available in Mac OS X v10.0 and later.

Declared in `AEDataModel.h`.

Version Notes

In Mac OS X version 10.4, you should use `typeUTF16ExternalRepresentation` or `typeUTF8Text` to represent text. In earlier versions of Mac OS X, the recommended text type is `typeUnicodeText`. All of the other constants in this enum are deprecated due to their lack of explicit encoding or byte order definition.

The implicitly encoded text types, `typeText`, `typeCString`, and `typePString`, are all deprecated in Mac OS X, because they are incapable of representing international characters and may be reinterpreted in unpredictable ways. Additionally, `typeCString` and `typePString` do not support the full range of text coercions, and will be removed entirely in a future release. `typeStyledText` and `typeIntlText`, while they have explicit encodings, are not recommended, since they are incapable of representing Unicode-only characters, such as Hungarian, Arabic, or Thai.

Result Codes

Because the Apple Event Manager uses the services of the Event Manager, the functions described in this document may return Event Manager result codes in addition to the Apple Event Manager result codes listed here. Less commonly, an Apple Event Manager function may return other result codes, including some of those found in the CarbonCore header file `MacErrors.h`.

For result codes for the AEBuild-related functions, see "AEBuild Error Codes" (page 179).

Result Code	Value	Description
`noPortErr`	-903	Client hasn't set `'SIZE'` resource to indicate awareness of high-level events Available in Mac OS X v10.0 and later.
`destPortErr`	-906	Server hasn't set `'SIZE'` resource to indicate awareness of high-level events, or else is not present Available in Mac OS X v10.0 and later.
`sessClosedErr`	-917	The `kAEDontReconnect` flag in the `sendMode` parameter was set and the server quit, then restarted Available in Mac OS X v10.0 and later.
`errAECoercionFail`	-1700	Data could not be coerced to the requested descriptor type Available in Mac OS X v10.0 and later.
`errAEDescNotFound`	-1701	Descriptor was not found Available in Mac OS X v10.0 and later.
`errAECorruptData`	-1702	Data in an Apple event could not be read Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEWrongDataType	-1703	Wrong descriptor type Available in Mac OS X v10.0 and later.
errAENotAEDesc	-1704	Not a valid descriptor Available in Mac OS X v10.0 and later.
errAEBadListItem	-1705	Operation involving a list item failed Available in Mac OS X v10.0 and later.
errAENewerVersion	-1706	Need a newer version of the Apple Event Manager Available in Mac OS X v10.0 and later.
errAENotAppleEvent	-1707	The event is not in AppleEvent format. Available in Mac OS X v10.0 and later.
errAEEventNotHandled	-1708	Event wasn't handled by an Apple event handler Available in Mac OS X v10.0 and later.
errAEReplyNotValid	-1709	AEResetTimer was passed an invalid reply Available in Mac OS X v10.0 and later.
errAEUnknownSendMode	-1710	Invalid sending mode was passed Available in Mac OS X v10.0 and later.
errAEWaitCanceled	-1711	User canceled out of wait loop for reply or receipt Available in Mac OS X v10.0 and later.
errAETimeout	-1712	Apple event timed out Available in Mac OS X v10.0 and later.
errAENoUserInteraction	-1713	No user interaction allowed Available in Mac OS X v10.0 and later.
errAENotASpecialFunction	-1714	Wrong keyword for a special function Available in Mac OS X v10.0 and later.
errAEParamMissed	-1715	A required parameter was not accessed. Available in Mac OS X v10.0 and later.
errAEUnknownAddressType	-1716	Unknown Apple event address type Available in Mac OS X v10.0 and later.
errAEHandlerNotFound	-1717	No handler found for an Apple event Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEReplyNotArrived	-1718	Reply has not yet arrived Available in Mac OS X v10.0 and later.
errAEIllegalIndex	-1719	Not a valid list index Available in Mac OS X v10.0 and later.
errAEImpossibleRange	-1720	The range is not valid because it is impossible for a range to include the first and last objects that were specified; an example is a range in which the offset of the first object is greater than the offset of the last object Available in Mac OS X v10.0 and later.
errAEWrongNumberArgs	-1721	The number of operands provided for the kAENOT logical operator is not 1 Available in Mac OS X v10.0 and later.
errAEAccessorNotFound	-1723	There is no object accessor function for the specified object class and container type Available in Mac OS X v10.0 and later.
errAENoSuchLogical	-1725	The logical operator in a logical descriptor is not kAEAND, kAEOR, or kAENOT Available in Mac OS X v10.0 and later.
errAEBadTestKey	-1726	The descriptor in a test key is neither a comparison descriptor nor a logical descriptor Available in Mac OS X v10.0 and later.
errAENotAnObjectSpec	–1727	The objSpecifier parameter of AEResolve is not an object specifier
errAENoSuchObject	-1728	Runtime resolution of an object failed. Available in Mac OS X v10.0 and later.
errAENegativeCount	-1729	An object-counting function returned a negative result Available in Mac OS X v10.0 and later.
errAEEmptyListContainer	-1730	The container for an Apple event object is specified by an empty list Available in Mac OS X v10.0 and later.
errAEUnknownObjectType	-1731	The object type isn't recognized Available in Mac OS X v10.0 and later.
errAERecordingIsAlreadyOn	-1732	Recording is already on Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errAEReceiveTerminate	-1733	Break out of all levels of AEReceive to the topmost (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEReceiveEscapeCurrent	-1734	Break out of lowest level only of AEReceive (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEEventFiltered	-1735	Event has been filtered and should not be propagated (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEDuplicateHandler	-1736	Attempt to install handler in table for identical class and ID (1.1 or greater) Available in Mac OS X v10.0 and later.
errAEStreamBadNesting	-1737	Nesting violation while streaming Available in Mac OS X v10.0 and later.
errAEStreamAlreadyConverted	-1738	Attempt to convert a stream that has already been converted Available in Mac OS X v10.0 and later.
errAEDescIsNull	-1739	Attempt to perform an invalid operation on a null descriptor Available in Mac OS X v10.0 and later.
errAEBuildSyntaxError	-1740	AEBuildDesc and related functions detected a syntax error Available in Mac OS X v10.0 and later.
errAEBufferTooSmall	-1741	Buffer for AEFlattenDesc too small Available in Mac OS X v10.0 and later.
errASCantConsiderAndIgnore	-2720	Can't both consider and ignore <attribute>. Available in Mac OS X v10.0 and later.
errASCantCompareMoreThan32k	-2721	Can't perform operation on text longer than 32K bytes. Available in Mac OS X v10.0 and later.
errASTerminologyNestingTooDeep	-2760	Tell statements are nested too deeply. Available in Mac OS X v10.0 and later.
errASIllegalFormalParameter	-2761	<name> is illegal as a formal parameter. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errASParameterNotForEvent	-2762	<name> is not a parameter name for the event <event>. Available in Mac OS X v10.0 and later.
errASNoResultReturned	-2763	No result was returned for some argument of this expression. Available in Mac OS X v10.0 and later.
errAEEventFailed	-10000	Apple event handler failed. Available in Mac OS X v10.0 and later.
errAETypeError	-10001	A descriptor type mismatch occurred. Available in Mac OS X v10.0 and later.
errAEBadKeyForm	-10002	Invalid key form. Available in Mac OS X v10.0 and later.
errAENotModifiable	-10003	Can't set <object or data> to <object or data>. Access not allowed. Available in Mac OS X v10.0 and later.
errAEPrivilegeError	-10004	A privilege violation occurred. Available in Mac OS X v10.0 and later.
errAEReadDenied	-10005	The read operation was not allowed. Available in Mac OS X v10.0 and later.
errAEWriteDenied	-10006	Can't set <object or data> to <object or data>. Available in Mac OS X v10.0 and later.
errAEIndexTooLarge	-10007	The index of the event is too large to be valid. Available in Mac OS X v10.0 and later.
errAENotAnElement	-10008	The specified object is a property, not an element. Available in Mac OS X v10.0 and later.
errAECantSupplyType	-10009	Can’t supply the requested descriptor type for the data. Available in Mac OS X v10.0 and later.
errAECantHandleClass	-10010	The Apple event handler can’t handle objects of this class. Available in Mac OS X v10.0 and later.
errAEInTransaction	-10011	Couldn’t handle this command because it wasn’t part of the current transaction. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
`errAENoSuchTransaction`	-10012	The transaction to which this command belonged isn't a valid transaction. Available in Mac OS X v10.0 and later.
`errAENoUserSelection`	-10013	There is no user selection. Available in Mac OS X v10.0 and later.
`errAENotASingleObject`	-10014	Handler only handles single objects. Available in Mac OS X v10.0 and later.
`errAECantUndo`	-10015	Can't undo the previous Apple event or user action. Available in Mac OS X v10.0 and later.
`errAENotAnEnumMember`	-10023	Enumerated value in `SetData` is not allowed for this property Available in Mac OS X v10.0 and later.
`errAECantPutThatThere`	-10024	In make new, duplicate, etc. class can't be an element of container Available in Mac OS X v10.0 and later.
`errAEPropertiesClash`	-10025	Illegal combination of properties settings for SetData, make new, or duplicate Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Apple Event Manager selectors defined in the Gestalt Manager. For more information see *Inside Mac OS X: Gestalt Manager Reference*.

Document Revision History

This table describes the changes to *Apple Event Manager Reference*.

Date	Notes
2007-07-13	Added and modified function and constant descriptions.
	These functions now have descriptions: `AEPutKeyDesc` (page 78), `AEPutKeyPtr` (page 79), `AEDeleteKeyDesc` (page 39), `AEGetKeyDesc` (page 52), `AEGetKeyPtr` (page 53), and `AESizeOfKeyDesc` (page 99).
	These constants now have descriptions: `keyAELaunchedAsLogInItem` and `keyAELaunchedAsServiceItem` in "`Launch Apple Event Constants`" (page 212); and `typeUInt16` and `typeUInt64` in "`Numeric Descriptor Type Constants`" (page 213).
	In "Descriptor Type Constants" (page 197), clarified description for `typeFileURL` and added note to Discussion section about working with Fast User Switching.
	Added Version Notes section to `AEResetTimer` (page 88), noting that prior to Mac OS X version 10.3, calling that function did not reset the timeout value.
	In the Discussion sections for `AEGetDescData` (page 48) and `AEGetDescDataSize` (page 50), noted that you can only use these functions with value descriptors created by `AECreateDesc` (page 33).
	For the functions in "Suspending and Resuming Apple Event Handling" (page 23), noted that they should be called only on the main thread.
	Added Special Considerations section for the function `AESendMessage` (page 94), describing a potential bug and providing a link to a sample code work-around.
	Added information to "Introduction to Apple Event Manager Reference" (page 13) about thread safety, about forcing a connection to the window server, and about the location of the AE framework (now a subframework of the CoreServices framework). Also added a link to *AppleScript Terminology and Apple Event Codes Reference*.
2006-09-05	Noted that AEFlattenDesc and AEUnflattenDesc require no developer steps with respect to the endianness of the serialized data.
2005-08-11	Revised descriptions for AESuspendTheCurrentEvent and AEResumeTheCurrentEvent.
2005-07-07	Added missing constant descriptions and fixed minor bugs.

Date	Notes
	Added enumeration "kTSMHiliteCaretPosition" (page 240), and noted that starting in Mac OS X version 10.4, you should use constants from that enumeration, rather than from "kCaretPosition" (page 231).
	In "Descriptor Type Constants" (page 197), added note that a descriptor of type `typeFileURL` doesn't represent a CFURL, it represents a C-string-style file path.
2005-04-29	Updated to cover a small number of changes for Mac OS X v10.4 and fix minor bugs.
	Added documentation for the constants in the enum "`typeUnicodeText`" (page 251), including the new constants `typeUTF16ExternalRepresentation` and `typeUTF8Text`. See important information in the Version Notes section.
	Added documentation for the constant `kAEOpenContents` in the section "Event ID Constants" (page 202). This constant is new in Mac OS X v10.4.
	Added documentation for the constant `keyAESearchText` in the enum "keyAEPropData" (page 235). This constant is new in Mac OS X v10.4.
	For a number of functions and data types, added links to related information in new document *Apple Events Programming Guide*.
	Added missing descriptions for the constants `typeFSRef` and `typeFileURL` in the section "Descriptor Type Constants" (page 197).
	Filled in missing descriptions for some constants in "Apple Event Manager Result Codes" (page 252).
	Added documentation for the constant `keyReplyRequestedAttr` in the section "Keyword Attribute Constants" (page 209).
	Made minor changes to introductory text in "Apple Event Manager Functions" (page 24).
	Added Version Notes sections to `AEBuildAppleEvent` (page 24)and `AEBuildDesc` (page 26), noting that prior to Mac OS X version 10.3, these functions would fail if you supplied a data parameter with size greater than 32767 bytes.
	Added Description section for "`typeApplicationBundleID`" (page 244).
	Deleted duplicate definition of constant `typeApplicationURL`, leaving only the one in "Descriptor Type Constants" (page 197).
	Noted that the functions `AEGetParamDesc` (page 59) and `AEGetParamPtr` (page 60) work with Apple event records (type `AERecord` (page 173)), as well as with Apple events (type `AppleEvent` (page 175)).
	Reduced use of the word "record," which often appeared gratuitously with data structures that were converted from Pascal record types long ago.

Date	Notes
2004-01-19	In the function call example in the description of the `AEEventSource` (page 170) typedef, changed the call to use `sizeof (AEEventSource)`.
2003-12-19	Added note on use of pointer-based and descriptor-based coercion handlers to `AECoercePtr` (page 30). This information applies to Mac OS X version 10.2 and later.
	Added a missing "&" to the function call example in the description of the `AEEventSource` (page 170) typedef.
2003-08-06	Added callback, constant, data type, and function descriptions for obtaining a list of remote processes with the remote process resolver mechanism.
	Added descriptions for these functions: `AECreateRemoteProcessResolver` (page 36), `AEDisposeRemoteProcessResolver` (page 40), `AERemoteProcessResolverGetProcesses` (page 82), `AERemoteProcessResolverScheduleWithRunLoop` (page 83)
	Added description for this callback: `AERemoteProcessResolverCallback` (page 148)
	Added descriptions for these data types: `AERemoteProcessResolverContext` (page 163), `AERemoteProcessResolverRef` (page 173)
	Added descriptions for these constants: `kAERemoteProcessURLKey` (page 218), `kAERemoteProcessNameKey` (page 218), `kAERemoteProcessUserIDKey` (page 218), `kAERemoteProcessProcessIDKey` (page 219), `typeApplicationURL` (page 200), `"typeKernelProcessID"` (page 245), `"typeMachPort"` (page 246)
	Added thread safety information for many Apple Event Manager functions.
	Reordered some constants that were not in alphabetical order.
2003-02-01	Updated formatting.

Document Revision History

Index

A

C

D

E

F

I

K

L

N

O

P

R

S

T

U

V

W