
Open Scripting Architecture Reference

[Carbon > Scripting & Automation](#)



2007-05-07



Apple Inc.
© 1993, 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

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

Apple, the Apple logo, AppleScript, Carbon, Cocoa, Mac, Mac OS, Macintosh, and OpenDoc are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY

DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Open Scripting Architecture Reference 9

Overview	9
Functions by Task	9
Saving and Loading Script Data	9
Executing and Disposing of Scripts	10
Setting and Getting Script Information	10
Manipulating the Active Function	10
Compiling Scripts	10
Getting Source Data	11
Coercing Script Values	11
Manipulating the Create and Send Functions	11
Recording Scripts	12
Executing Scripts in One Step	12
Copying a Scripting Dictionary as a Scripting Definition File	12
Manipulating Dialects	13
Using Script Contexts to Handle Apple Events	13
Initializing AppleScript	13
Getting and Setting Styles for Source Data	14
Getting and Setting the Default Scripting Component	14
Using Component-Specific Routines	14
Manipulating Trailers for Generic Storage Descriptor Records	14
Miscellaneous	15
Creating, Invoking and Disposing Universal Procedure Pointers	15
Deprecated Functions	16
Functions	17
ASCopySourceAttributes	17
ASGetAppTerminology	18
ASGetHandler	18
ASGetProperty	19
ASGetSourceStyleNames	19
ASInit	20
ASSetHandler	21
ASSetProperty	22
ASSetSourceAttributes	22
DisposeOSAActiveUPP	23
DisposeOSACreateAppleEventUPP	23
DisposeOSASendUPP	23
InvokeOSAActiveUPP	24
InvokeOSACreateAppleEventUPP	24
InvokeOSASendUPP	25
NewOSAActiveUPP	25

NewOSACreateAppleEventUPP	25
NewOSASendUPP	26
OSAAddStorageType	26
OSAAvailableDialectCodeList	27
OSAAvailableDialects	27
OSACoerceFromDesc	28
OSACoerceToDesc	29
OSACompile	30
OSACompileExecute	31
OSACopyDisplayString	32
OSACopyID	33
OSACopyScriptingDefinition	33
OSACopySourceString	34
OSADebuggerCreateSession	35
OSADebuggerDisposeCallFrame	35
OSADebuggerDisposeSession	36
OSADebuggerGetBreakpoint	36
OSADebuggerGetCallFrameState	37
OSADebuggerGetCurrentCallFrame	37
OSADebuggerGetDefaultBreakpoint	37
OSADebuggerGetPreviousCallFrame	38
OSADebuggerGetSessionState	38
OSADebuggerGetStatementRanges	39
OSADebuggerGetVariable	39
OSADebuggerSessionStep	39
OSADebuggerSetBreakpoint	40
OSADebuggerSetVariable	40
OSADisplay	41
OSADispose	42
OSADoEvent	42
OSADoScript	44
OSADoScriptFile	45
OSAExecute	46
OSAExecuteEvent	47
OSAGenericToRealID	48
OSAGetActiveProc	49
OSAGetCreateProc	50
OSAGetCurrentDialect	50
OSAGetDefaultScriptingComponent	51
OSAGetDialectInfo	51
OSAGetHandler	52
OSAGetHandlerNames	53
OSAGetProperty	54
OSAGetPropertyNames	55
OSAGetResumeDispatchProc	56
OSAGetScriptInfo	56

OSAGetScriptingComponent	57
OSAGetScriptingComponentFromStored	58
OSAGetSendProc	59
OSAGetSource	59
OSAGetStorageType	60
OSAGetSysTerminology	61
OSALoad	62
OSALoadExecute	63
OSALoadExecuteFile	64
OSALoadFile	65
OSAMakeContext	66
OSARealToGenericID	66
OSARemoveStorageType	67
OSAScriptError	68
OSAScriptingComponentName	69
OSASetActiveProc	69
OSASetCreateProc	70
OSASetCurrentDialect	71
OSASetDefaultScriptingComponent	71
OSASetDefaultTarget	72
OSASetHandler	73
OSASetProperty	73
OSASetResumeDispatchProc	74
OSASetScriptInfo	75
OSASetSendProc	76
OSAStartRecording	77
OSAStopRecording	78
OSAStore	79
OSAStoreFile	80
Callbacks	81
OSAActiveProcPtr	81
OSACreateAppleEventProcPtr	81
OSASendProcPtr	83
Data Types	84
OSAID	84
GenericID	85
OSAError	85
ScriptingComponentSelector	85
StatementRange	85
OSAActiveUPP	86
OSACreateAppleEventUPP	86
OSASendUPP	86
OSADebugCallFrameRef	86
OSADebugSessionRef	87
Constants	88
cClosure	88

- cCoercion 88
- cHandleBreakpoint 88
- Component Flags 89
- Considerations Flags 90
- Considerations Bit Masks 91
- cString 92
- Current Dialect Constants 92
- Date and Time Constants 92
- Default Initialization Values 92
- Dialect Descriptor Constants 94
- Generic Scripting Component Selectors 95
- Global Properties 95
- kASAdd 95
- kASAnd 95
- kASErrorEventCode 96
- kASStartLogEvent 96
- kDialectBundleResType 96
- keyAETarget 97
- keyAppHandledCoercion 97
- keyASPrepositionAt 97
- keyASPrepositionOver 98
- keyOSASourceEnd 98
- keyOSASourceStart 98
- keyProcedureName 99
- keyProgramState 99
- kGenericComponentVersion 99
- kOSAComponentType 100
- kOSAGenericScriptingComponentSubtype 100
- kOSAModeDontDefine 100
- kOSANullScript 100
- kOSARecordedText 101
- kOSAScriptResourceType 101
- kOSASelectComponentSpecificStart 101
- kOSASelectCopyScript 102
- kOSASuite 102
- Mode Flags 102
- Null Mode Flags 106
- OSADebugStepKind 106
- OSAProgramState 106
- OSAScriptError Selectors 106
- Recording Constants 108
- Resume Dispatch Function Constants 108
- Script Document File Type 109
- Script Information Selectors 109
- Source Constants 111
- Source Style Constants 111

typeAppleScript 112
typeOSAErrorRange 113
typeOSAGenericStorage 113
typeStatementRange 114
Weekdays 114
Result Codes 114

Appendix A **Deprecated Open Scripting Architecture Functions 119**

Deprecated in Mac OS X v10.5 119
ASGetSourceStyles 119
ASSetSourceStyles 119
OSAGetAppTerminology 120

Document Revision History 123

Index 125

Open Scripting Architecture Reference

Framework:	Carbon/Carbon.h
Declared in	ASDebugging.h ASRegistry.h AppleScript.h OSA.h OSAComp.h OSAGeneric.h

Overview

The Open Scripting Architecture (OSA) provides a standard and extensible mechanism for interapplication communication in Mac OS X. It provides support for creating scriptable applications and for writing scripting components to implement scripting languages. Every Mac OS X system includes the AppleScript component, which implements AppleScript, the standard scripting language defined by Apple. However, developers can write scripting components for additional scripting languages. For conceptual information on the OSA, see “Open Scripting Architecture” in [AppleScript Overview](#).

You need to use this reference if you are writing a scripting component or if your application needs to interact with scripting components to manipulate and execute scripts. The API described in this document is implemented by the OpenScripting framework, a subframework of the Carbon framework. For information about working with components, see [Scripting Components](#) in [Inside Macintosh: Interapplication Communication](#).

Important: Do not rely on the API descriptions in [Interapplication Communication—Open Scripting Architecture Reference](#) provides the current API documentation.

The Apple Event Manager, another part of the OSA, is implemented primarily by the AE framework, a subframework of the Application Services framework, and is documented in [Apple Event Manager Reference](#) and [Apple Events Programming Guide](#). Applications use the Apple Event Manager to send and respond to Apple events and to make their operations and data available to AppleScript scripts.

Functions by Task

Saving and Loading Script Data

[OSALoad](#) (page 62)

Loads script data.

[OSALoadFile](#) (page 65)

Loads a script from the specified file into the specified scripting component, compiling the script if the file is a text file.

[OSAStore](#) (page 79)

Gets a handle to script data in the form of a storage descriptor record.

[OSAStoreFile](#) (page 80)

Stores a script into the specified file.

Executing and Disposing of Scripts

To execute a script, your application must first obtain a valid script ID for a compiled script or script context. You can use either the `OSALoad` function or the optional `OSACompile` function to obtain a script ID.

[OSAExecute](#) (page 46)

Executes a compiled script or a script context.

[OSAScriptError](#) (page 68)

Gets information about errors that occur during script execution.

[OSADispose](#) (page 42)

Reclaims the memory occupied by script data.

Setting and Getting Script Information

[OSASetScriptInfo](#) (page 75)

Sets information about script data according to the value you pass in the selector parameter.

[OSAGetScriptInfo](#) (page 56)

Obtains information about script data according to the value you pass in the selector parameter.

Manipulating the Active Function

[OSASetActiveProc](#) (page 69)

Sets the active function that a scripting component calls periodically while executing a script.

[OSAGetActiveProc](#) (page 49)

Gets a pointer to the active function that a scripting component is currently using.

Compiling Scripts

Scripting components can provide three optional functions that get the name of a scripting component, compile a script, and update a script ID. A scripting component that supports the functions in this section has the `kOSASupportsCompiling` bit set in the `componentFlags` field of its component description record.

[OSAScriptingComponentName](#) (page 69)

Gets the name of a scripting component.

[OSACompile](#) (page 30)

Compiles the source data for a script and obtain a script ID for a compiled script or a script context.

[OSACopyID](#) (page 33)

Updates script data after editing or recording and to perform undo or revert operations on script data.

Getting Source Data

[OSAGetSource](#) (page 59)

Decompiles the script data identified by a script ID and obtains the equivalent source data.

[OSADisplay](#) (page 41)

Converts a script value to text. Your application can then use its own functions to display this text to the user.

[OSACopyDisplayString](#) (page 32)

Converts a script value to an attributed Unicode text string, which your application can display to the user.

[OSACopySourceString](#) (page 34)

Decompiles the script data for the specified script and returns a copy of the equivalent source data as an attributed Unicode text string.

Coercing Script Values

Scripting components can provide support for two optional functions which coerce data in a descriptor record to a script value and coerce a script value to data in a descriptor record. A scripting component that supports the functions in this section has the `kOSASupportsAECOercion` bit set in the `componentFlags` field of its component description record.

[OSACoerceFromDesc](#) (page 28)

Obtains the script ID for a script value that corresponds to the data in a descriptor record.

[OSACoerceToDesc](#) (page 29)

Coerces a script value to a descriptor record of a desired descriptor type.

Manipulating the Create and Send Functions

Some scripting components provide functions that allow your application to set or get pointers to the create and send functions used by the scripting component when it sends and creates Apple events during script execution. If you do not set the pointers that specify these functions, the scripting component uses the standard `AECreatAppleEvent` and `AESend` functions with default parameters. A scripting component that supports the functions described in this section has the `kOSASupportsAESending` bit set in the `componentFlags` field of its component description record.

[OSASetCreateProc](#) (page 70)

Specifies a create function that a scripting component should use instead of the Apple Event Manager's `AECreatAppleEvent` function when creating Apple events.

[OSAGetCreateProc](#) (page 50)

Gets a pointer to the create function that a scripting component is currently using to create Apple events.

[OSASetSendProc](#) (page 76)

Specifies a send function that a scripting component should use instead of the Apple Event Manger's `AESend` function when sending Apple events.

[OSAGetSendProc](#) (page 59)

Gets a pointer to the send function that a scripting component is currently using.

[OSASetDefaultTarget](#) (page 72)

Sets the default target application for Apple events.

Recording Scripts

Script editors use these functions to allow users to control recording. Any application can use these functions to provide its own script-recording interface. A scripting component that supports the functions described in this section has the `kOSASupportsRecording` bit set in the `componentFlags` field of its component description record.

[OSAStartRecording](#) (page 77)

Turns on Apple event recording and records subsequent Apple events in a compiled script.

[OSAStopRecording](#) (page 78)

Turns off Apple event recording.

Executing Scripts in One Step

You can use these functions if you know that the script data to be executed will be executed only once. A scripting component that supports the functions described in this section has the `kOSASupportsConvenience` bit set in the `componentFlags` field of its component description record.

[OSACompileExecute](#) (page 31)

Compiles and executes a script in a single step rather than calling `OSACompile` and `OSAExecute`.

[OSADoScript](#) (page 44)

Compiles and executes a script and converts the resulting script value to text in a single step rather than calling `OSACompile`, `OSAExecute`, and `OSADisplay`.

[OSADoScriptFile](#) (page 45)

Loads a script from the specified file, compiles the script if the file is a text file, executes the script, converts the resulting script value to text, and stores the script back into the file if the script has persistent properties and the file is not a text file.

[OSALoadExecute](#) (page 63)

Loads and executes a script in a single step rather than calling `OSALoad` and `OSAExecute`.

[OSALoadExecuteFile](#) (page 64)

Loads a script from the specified file into the specified scripting component, compiles the script if the file is a text file, and executes the script.

Copying a Scripting Dictionary as a Scripting Definition File

[OSACopyScriptingDefinition](#) (page 33)

Creates a copy of a scripting definition (sdef) from the specified file or bundle.

Manipulating Dialects

Scripting components that provide several dialects may provide five functions that allow you to switch between dialects dynamically and get information about currently available dialects. The codes for specific dialects are provided by the scripting component. A scripting component that supports the functions described in this section has the `kOSASupportsDialects` bit set in the `componentFlags` field of its component description record.

[OSASetCurrentDialect](#) (page 71)

Sets the current dialect for a scripting component.

[OSAGetCurrentDialect](#) (page 50)

Gets the dialect code for the dialect currently being used by a scripting component.

[OSAAvailableDialectCodeList](#) (page 27)

Obtains a descriptor list containing dialect codes for each of a scripting component's currently available dialects.

[OSAGetDialectInfo](#) (page 51)

Gets information about a specified dialect provided by a specified scripting component.

[OSAAvailableDialects](#) (page 27)

Obtains a descriptor list containing information about each of the currently available dialects for a scripting component.

Using Script Contexts to Handle Apple Events

The optional functions described in this section allow your application to use script contexts to handle Apple events. One way to do this is to install a general Apple event handler in your application's special handler dispatch table. The general Apple event handler provides initial handling for every Apple event received by your application. A scripting component that supports the functions described in this section has the `kOSASupportsEventHandling` bit set in the `componentFlags` field of its component description record.

[OSASetResumeDispatchProc](#) (page 74)

Sets the resume dispatch function called by a scripting component during execution of an AppleScript `continue` statement or its equivalent.

[OSAGetResumeDispatchProc](#) (page 56)

Gets the resume dispatch function currently being used by a scripting component instance during execution of an AppleScript `continue` statement or its equivalent

[OSAExecuteEvent](#) (page 47)

Handles an Apple event with the aid of a script context and obtains a script ID for the resulting script value.

[OSADoEvent](#) (page 42)

Handles an Apple event with the aid of a script context and obtains a reply event.

[OSAMakeContext](#) (page 66)

Gets a script ID for a new script context.

Initializing AppleScript

[ASInit](#) (page 20)

Initializes the AppleScript component.

Getting and Setting Styles for Source Data

[ASCopySourceAttributes](#) (page 17)

Gets the current text style attributes AppleScript uses to display script text.

[ASSetSourceAttributes](#) (page 22)

Sets the text style attributes used by the AppleScript component to display scripts.

[ASGetSourceStyleNames](#) (page 19)

Obtains a list of style names that are each formatted according to the script format styles currently used by the AppleScript component.

Getting and Setting the Default Scripting Component

The default scripting component for any instance of the generic scripting component is initially AppleScript, but you can change it if necessary.

[OSAGetDefaultScriptingComponent](#) (page 51)

Gets the subtype code for the default scripting component associated with an instance of the generic scripting component.

[OSASetDefaultScriptingComponent](#) (page 71)

Sets the default scripting component associated with an instance of the generic scripting component.

Using Component-Specific Routines

You can't use the generic scripting component to call a component-specific routine. Instead, you must use an instance of the specific scripting component that supports the routine.

To facilitate the use of component-specific routines, the generic scripting component allows you to identify the scripting component that created stored script data, get an instance of a specified scripting component, and convert between generic script IDs and component-specific script IDs.

[OSAGetScriptingComponentFromStored](#) (page 58)

Gets the subtype code for a scripting component that created a storage descriptor record.

[OSAGetScriptingComponent](#) (page 57)

Gets the instance of a scripting component for a specified subtype.

[OSAGenericToRealID](#) (page 48)

Converts a generic script ID to the corresponding component-specific script ID.

[OSARealToGenericID](#) (page 66)

Converts a component-specific script ID to the corresponding generic script ID.

Manipulating Trailers for Generic Storage Descriptor Records

All scripting components must use the `OSAGetStorageType`, `OSAAddStorageType`, and `OSARemoveStorageType` functions described in this section to add, remove, and inspect the trailers appended to script data in generic storage descriptor records.

[OSAGetStorageType](#) (page 60)

Retrieves the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

[OSAAddStorageType](#) (page 26)

Adds a trailer to the script data in a generic storage descriptor record.

[OSARemoveStorageType](#) (page 67)

Removes a trailer from the script data in a generic storage descriptor record

Miscellaneous

[ASGetAppTerminology](#) (page 18)

Deprecated. Use [OSAGetAppTerminology](#) (page 120) instead.

[ASGetHandler](#) (page 18)

Deprecated. Use [OSAGetHandler](#) (page 52) instead.

[ASGetProperty](#) (page 19)

Deprecated. Use [OSAGetProperty](#) (page 54) instead.

[ASSetHandler](#) (page 21)

Deprecated. Use [OSASetHandler](#) (page 73) instead.

[ASSetProperty](#) (page 22)

Deprecated. Use [OSASetProperty](#) (page 73) instead.

[OSAGetHandler](#) (page 52)

Gets a script ID for the specified script handler from the specified script.

[OSAGetHandlerNames](#) (page 53)

Gets a list of all handler names in the specified script as an `AEDescList` of descriptors of type `typeChar`.

[OSAGetProperty](#) (page 54)

Gets the value of a specified script property from a specified script.

[OSAGetPropertyNames](#) (page 55)

Gets a list of all property names from the specified script.

[OSAGetSysTerminology](#) (page 61)

Gets one or more scripting terminology resources from the OSA system.

[OSASetHandler](#) (page 73)

Sets a specified script handler in the specified script to the supplied handler.

[OSASetProperty](#) (page 73)

Sets the value of a script property in a specified script, creating the property if it does not already exist.

[OSAGetAppTerminology](#) (page 120) **Deprecated in Mac OS X v10.5**

Gets one or more scripting terminology resources from the specified file. (**Deprecated**. Use [OSACopyScriptingDefinition](#) (page 33) instead.)

Creating, Invoking and Disposing Universal Procedure Pointers

[NewOSAActiveUPP](#) (page 25)

Creates a new universal procedure pointer to an application-defined active function.

[NewOSACreateAppleEventUPP](#) (page 25)

Creates a new universal procedure pointer to an application-defined Apple event creation function.

[NewOSASendUPP](#) (page 26)

Creates a new universal procedure pointer to an application-defined send function.

[DisposeOSAActiveUPP](#) (page 23)

Disposes of a universal procedure pointer to an application-defined active function.

[DisposeOSACreateAppleEventUPP](#) (page 23)

Disposes of a universal procedure pointer to an application-defined Apple event create function.

[DisposeOSASendUPP](#) (page 23)

Disposes of a universal procedure pointer to an application-defined send function.

[InvokeOSAActiveUPP](#) (page 24)

Invokes an application-defined active function.

[InvokeOSACreateAppleEventUPP](#) (page 24)

Invokes an application-defined Apple event creation function.

[InvokeOSASendUPP](#) (page 25)

Invokes an application-defined send function.

Deprecated Functions



Warning: Do not use the OSA debugging functions listed here. They were were not intended for public use, they do not work, and they will return an error.

[OSADebuggerCreateSession](#) (page 35)

Do not use.

[OSADebuggerDisposeCallFrame](#) (page 35)

Do not use.

[OSADebuggerDisposeSession](#) (page 36)

Do not use.

[OSADebuggerGetBreakpoint](#) (page 36)

Do not use.

[OSADebuggerGetCallFrameState](#) (page 37)

Do not use.

[OSADebuggerGetCurrentCallFrame](#) (page 37)

Do not use.

[OSADebuggerGetDefaultBreakpoint](#) (page 37)

Do not use.

[OSADebuggerGetPreviousCallFrame](#) (page 38)

Do not use.

[OSADebuggerGetSessionState](#) (page 38)

Do not use.

[OSADebuggerGetStatementRanges](#) (page 39)

Do not use.

[OSADebuggerGetVariable](#) (page 39)

Do not use.

[OSADebuggerSessionStep](#) (page 39)

Do not use.

[OSADebuggerSetBreakpoint](#) (page 40)

Do not use.

[OSADebuggerSetVariable](#) (page 40)

Do not use.

[ASGetSourceStyles](#) (page 119) **Deprecated in Mac OS X v10.5**

Gets the script format styles currently used by the AppleScript component to display scripts. **(Deprecated.** Use [ASGetSourceStyleNames](#) (page 19) instead.)

[ASSetSourceStyles](#) (page 119) **Deprecated in Mac OS X v10.5**

Sets the script format styles used by the AppleScript component to display scripts. **(Deprecated.** Use [ASSetSourceAttributes](#) (page 22) instead.)

Functions

ASCopySourceAttributes

Gets the current text style attributes AppleScript uses to display script text.

```
OSAError ASCopySourceAttributes (
    ComponentInstance scriptingComponent,
    CFArrayRef *resultingSourceAttributes
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingSourceAttributes

If successful, returns a reference to an array (of type `CFArray`) of dictionaries (of type `CFDictionary`) of text style attributes; otherwise, returns `nil`.

The order of elements in the array corresponds to the constants defined in “[Source Style Constants](#)” (page 111), and therefore also to the names returned by [ASGetSourceStyleNames](#) (page 19). For example, the first dictionary in the array (at position `kASSourceStyleUncompiledText`) describes the style for uncompiled text. However, you should not rely on there being any specific number of dictionaries in the returned array—instead, count the number of items in the array before accessing any of them.

This array is a copy and the caller is responsible for releasing it, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

A text style attribute is typically something that is meaningful to a `CFAttributedString`, such as the one returned by [OSACopyDisplayString](#) (page 32) or [OSACopySourceString](#) (page 34). However, clients may add other attributes using [ASSetSourceAttributes](#) (page 22).

Availability

Available in Mac OS X v10.5 and later.

Declared In

AppleScript.h

ASGetAppTerminology

Deprecated. Use [OSAGetAppTerminology](#) (page 120) instead.

```
OSAEError ASGetAppTerminology (  
    ComponentInstance scriptingComponent,  
    FSSpec *fileSpec,  
    short terminologyID,  
    Boolean *didLaunch,  
    AEDesc *terminologyList  
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Version Notes

Provided for backward compatibility only. Use [OSAGetAppTerminology](#) (page 120) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

ASDebugging.h

ASGetHandler

Deprecated. Use [OSAGetHandler](#) (page 52) instead.

```
OSAEError ASGetHandler (  
    ComponentInstance scriptingComponent,  
    OSAID contextID,  
    const AEDesc *handlerName,  
    OSAID *resultingCompiledScriptID  
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Version Notes

Provided for backward compatibility only. Use [OSAGetHandler](#) (page 52) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

ASDebugging.h

ASGetProperty

Deprecated. Use [OSAGetProperty](#) (page 54) instead.

```

OSAEError ASGetProperty (
    ComponentInstance scriptingComponent,
    OSAID contextID,
    const AEDesc *variableName,
    OSAID *resultingScriptValueID
);

```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Version Notes

Provided for backward compatibility only. Use [OSAGetProperty](#) (page 54) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

ASDebugging.h

ASGetSourceStyleNames

Obtains a list of style names that are each formatted according to the script format styles currently used by the AppleScript component.

```

OSAEError ASGetSourceStyleNames (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    AEDescList *resultingSourceStyleNamesList
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

modeFlags

Reserved for future use. Set to `kOSAModeNull`.

resultingSourceStyleNames

A pointer to a list of style names (for example, “Uncompiled Text,” “Normal Text”) that are each formatted according to the current script format styles. The order of the names corresponds to the order of the source style constants listed in [“Source Style Constants”](#) (page 111). For example, the first name in the list (at position `kASSourceStyleUncompiledText`) is formatted according to the style for uncompiled text.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

AppleScript.h

ASInit

Initializes the AppleScript component.

```

OSAEError ASInit (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    UInt32 minStackSize,
    UInt32 preferredStackSize,
    UInt32 maxStackSize,
    UInt32 minHeapSize,
    UInt32 preferredHeapSize,
    UInt32 maxHeapSize
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

modeFlags

Reserved for future use. Set to `kOSAModeNull`.

minStackSize

The minimum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

preferredStackSize

The preferred size for the portion of the application's heap used by the AppleScript component's application-specific stack.

maxStackSize

The maximum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

minHeapSize

The minimum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

preferredHeapSize

The preferred size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

maxHeapSize

The maximum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Your application should set the `modeFlags` parameter to `kOSAModeNull`. You can use the other parameters to specify memory sizes for the portion of your application's heap used by the AppleScript component for its application-specific heap and stack. If your application sets any of these parameters to 0, the AppleScript component uses the corresponding value in your application's 'scsz' resource. If that value is also set to 0, the AppleScript component uses the default values described in “[Default Initialization Values](#)” (page 92).

If your application doesn't call `ASInit` explicitly, the AppleScript component initializes itself using the values specified in your application's 'scsz' resource when your application first calls any scripting component routine. If any of these values are set to 0, the AppleScript component uses the corresponding default value.

If your application doesn't call `ASInit` explicitly and doesn't call any scripting component routines, the AppleScript component will not be initialized. For example, if your application opens and closes the AppleScript component or calls Component Manager functions such as `OpenDefaultComponent` or `FindNextComponent` but doesn't call any scripting component routines, the AppleScript component is not initialized.

When the AppleScript component is initialized, it uses your application's high memory to create the blocks that it locks for its own use. If you expect to lock any portion of high memory for a shorter time than you expect the AppleScript component to be available, you should call `ASInit` explicitly.

Version Notes

Starting in Mac OS X version 10.5, heap size parameter values are ignored—AppleScript's heap will grow as large as needed.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`AppleScript.h`

ASSetHandler

Deprecated. Use [OSASetHandler](#) (page 73) instead.

```
OSAEError ASSetHandler (
    ComponentInstance scriptingComponent,
    OSAID contextID,
    const AEDesc *handlerName,
    OSAID compiledScriptID
);
```

Return Value

A result code. See “[Result Codes](#)” (page 114).

Version Notes

Provided for backward compatibility only. Use [OSASetHandler](#) (page 73) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`ASDebugging.h`

ASSetProperty

Deprecated. Use [OSASetProperty](#) (page 73) instead.

```

OSAEError ASSetProperty (
    ComponentInstance scriptingComponent,
    OSAID contextID,
    const AEDesc *variableName,
    OSAID scriptValueID
);

```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Version Notes

Provided for backward compatibility only. Use [OSASetProperty](#) (page 73) instead.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

ASDebugging.h

ASSetSourceAttributes

Sets the text style attributes used by the AppleScript component to display scripts.

```

OSAEError ASSetSourceAttributes (
    ComponentInstance scriptingComponent,
    CFArrayRef sourceAttributes
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceAttributes

A reference to an array (of type `CFArray`) of dictionaries (of type `CFDictionary`) of text style attributes.

You can pass a `nil` reference for this parameter if you want the AppleScript component to display script text using its default styles.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

A text style attribute is typically something that is meaningful to a `CFAttributedString`, such as the one returned by [OSACopyDisplayString](#) (page 32) or [OSACopySourceString](#) (page 34). However, clients may add any attributes they like. Because of this, you should generally call `ASSetSourceAttributes` with a modified copy of the result from [ASCopySourceAttributes](#) (page 17), not a built-from-scratch set of attributes.

The order of elements in the array should correspond to the constants defined in “[Source Style Constants](#)” (page 111), and therefore also to the names returned by [ASGetSourceStyleNames](#) (page 19). After calling `ASSetSourceAttributes`, you must dispose of the style element array you used to specify the text style attributes.

Availability

Available in Mac OS X v10.5 and later.

Declared In

AppleScript.h

DisposeOSAActiveUPP

Disposes of a universal procedure pointer to an application-defined active function.

```
void DisposeOSAActiveUPP (
    OSAActiveUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

DisposeOSACreateAppleEventUPP

Disposes of a universal procedure pointer to an application-defined Apple event create function.

```
void DisposeOSACreateAppleEventUPP (
    OSACreateAppleEventUPP userUPP
);
```

Parameters

userUPP

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

DisposeOSASendUPP

Disposes of a universal procedure pointer to an application-defined send function.

```
void DisposeOSASendUPP (
    OSASendUPP userUPP
);
```

Parameters*userUPP*

The UPP to dispose of.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

InvokeOSAActiveUPP

Invokes an application-defined active function.

```
OSErr InvokeOSAActiveUPP (
    SRefCon refCon,
    OSAActiveUPP userUPP
);
```

Return ValueA result code. See [“Result Codes”](#) (page 114).**Availability**

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

InvokeOSACreateAppleEventUPP

Invokes an application-defined Apple event creation function.

```
OSErr InvokeOSACreateAppleEventUPP (
    AEEEventClass theAEEEventClass,
    AEEEventID theAEEEventID,
    const AEAddressDesc *target,
    short returnID,
    SInt32 transactionID,
    AppleEvent *result,
    SRefCon refCon,
    OSACreateAppleEventUPP userUPP
);
```

Return ValueA result code. See [“Result Codes”](#) (page 114).**Availability**

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

InvokeOSASendUPP

Invokes an application-defined send function.

```

OSErr InvokeOSASendUPP (
    const AppleEvent *theAppleEvent,
    AppleEvent *reply,
    AESendMode sendMode,
    AESendPriority sendPriority,
    SInt32 timeOutInTicks,
    AEIdleUPP idleProc,
    AEFilterUPP filterProc,
    SRefCon refCon,
    OSASendUPP userUPP
);

```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

NewOSAActiveUPP

Creates a new universal procedure pointer to an application-defined active function.

```

OSAActiveUPP NewOSAActiveUPP (
    OSAActiveProcPtr userRoutine
);

```

Parameters

userRoutine

A pointer to the active function.

Return Value

The new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

NewOSACreateAppleEventUPP

Creates a new universal procedure pointer to an application-defined Apple event creation function.

```
OSACreateAppleEventUPP NewOSACreateAppleEventUPP (
    OSACreateAppleEventProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to the creation function.

Return Value

The new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

NewOSASendUPP

Creates a new universal procedure pointer to an application-defined send function.

```
OSASendUPP NewOSASendUPP (
    OSASendProcPtr userRoutine
);
```

Parameters

userRoutine

A pointer to the send function.

Return Value

The new UPP.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAAddStorageType

Adds a trailer to the script data in a generic storage descriptor record.

```
OSErr OSAAddStorageType (
    AEDataStorage scriptData,
    DescType dscType
);
```

Parameters

scriptData

A handle to the script data.

dscType

The descriptor type to be specified in the trailer added to the script data.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAAddStorageType` function attaches a trailer to a handle (consequently expanding the data to which the handle refers) or updates an existing trailer.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAComp.h`

OSAAvailableDialectCodeList

Obtains a descriptor list containing dialect codes for each of a scripting component's currently available dialects.

```
OSAEError OSAAvailableDialectCodeList (
    ComponentInstance scriptingComponent,
    AEDesc *resultingDialectCodeList
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingDialectCodeList

A pointer to the returned descriptor list.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Each item in the descriptor list returned by `OSAAvailableDialectCodeList` is a descriptor record of descriptor type `typeInteger` containing a dialect code for one of the specified scripting component's currently available dialects. Dialect codes are defined by individual scripting components.

You can pass any dialect code you obtain using `OSAAvailableDialectCodeList` to `OSAGetDialectInfo` to get information about the corresponding dialect.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAAvailableDialects

Obtains a descriptor list containing information about each of the currently available dialects for a scripting component.

```

OSAEError OSAAvailableDialects (
    ComponentInstance scriptingComponent,
    AEDesc *resultingDialectInfoList
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingDialectInfoList

A pointer to the returned descriptor list.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Each item in the list returned by `OSAAvailableDialects` is an AE record of descriptor type `typeOSADialectInfo`. Each descriptor record in the descriptor list contains, at a minimum, four keyword-specified descriptor records with the keywords described in [“Dialect Descriptor Constants”](#) (page 94).

Rather than calling `OSAAvailableDialects` to obtain complete dialect information for a scripting component, it is usually more convenient to call `OSAAvailableDialectCodeList` to get a list of codes for a scripting component’s dialects, then call `OSAGetDialectInfo` to get information about the specific dialect you’re interested in.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACoerceFromDesc

Obtains the script ID for a script value that corresponds to the data in a descriptor record.

```

OSAEError OSACoerceFromDesc (
    ComponentInstance scriptingComponent,
    const AEDesc *scriptData,
    SInt32 modeFlags,
    OSAID *resultingScriptID
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to a descriptor record containing the script data to be coerced.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. If the *scriptData* parameter contains an Apple event, you can use any of the mode flags listed in [“Mode Flags”](#) (page 102).

resultingScriptValueID

A pointer to the resulting script ID for a script value. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

The `OSACoerceFromDesc` function coerces the descriptor record in the *scriptData* parameter to the equivalent script value and returns a script ID for that value.

If you pass `OSACoerceFromDesc` an Apple event in the *scriptData* parameter, it returns a script ID for the equivalent compiled script in the *resultingScriptValueID* parameter. In this case you can specify any of the *modeFlags* values used by `OSACompile` to control the way the compiled script is executed.

If you call `OSACoerceFromDesc` using an instance of the generic scripting component, the generic scripting component uses the default scripting component to perform the coercion.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACoerceToDesc

Coerces a script value to a descriptor record of a desired descriptor type.

```
OSAEError OSACoerceToDesc (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    DescType desiredType,
    SInt32 modeFlags,
    AEDesc *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script value to coerce. See the [OSAID](#) (page 84) data type.

desiredType

The desired descriptor type of the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`.

result

A pointer to the resulting descriptor record.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

The `OSACoerceToDesc` function coerces the script value identified by `scriptValueID` to a descriptor record of the type specified by the `desiredType` parameter, if possible. Valid types include all the standard descriptor types, plus any special types supported by the scripting component.

If you want the descriptor type of the descriptor record returned in the result parameter to be the same as the descriptor type returned by a scripting component, use `OSACoerceToDesc` and specify `typeWildcard` as the desired type. If you want to get a script value in a form that you can display for humans to read, use `OSADisplay`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACompile

Compiles the source data for a script and obtain a script ID for a compiled script or a script context.

```
OSAEError OSACompile (
    ComponentInstance scriptingComponent,
    const AEDesc *sourceData,
    SInt32 modeFlags,
    OSAID *previousAndResultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record containing suitable source data for the specified scripting component.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in “[Mode Flags](#)” (page 102).

previousAndResultingScriptID

A pointer to the script ID for the resulting compiled script. If the value of this parameter on input is `kOSANullScript`, `OSACompile` returns a new script ID for the compiled script data. If the value of this parameter on input is an existing script ID, `OSACompile` updates the script ID so that it refers to the newly compiled script data. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

You can pass a descriptor record containing source data suitable for a specific scripting component (usually text) to the `OSACompile` function to obtain a script ID for the equivalent compiled script or script context. To compile the source data as a script context for use with `OSAExecuteEvent` or `OSADoEvent`, you must set the `kOSAModeCompileIntoContext` flag, and the source data should include appropriate handlers.

After you have successfully compiled the script, you can use the returned script ID to refer to the compiled script when you call `OSAExecute` and other scripting component routines.

If you use `OSACompile` with an instance of the generic scripting component and pass `kOSANullScript` in the *previousAndResultingScriptID* parameter, the generic scripting component uses the default scripting component to compile the script.

If you're recompiling a script, specify the original script ID in the *previousAndResultingScriptID* parameter. The generic scripting component uses the script ID to determine which scripting component it should use to compile the script.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACompileExecute

Compiles and executes a script in a single step rather than calling `OSACompile` and `OSAExecute`.

```
OSALong OSACompileExecute (
    ComponentInstance scriptingComponent,
    const AEDesc *sourceData,
    OSAID contextID,
    SInt32 modeFlags,
    OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record identifying suitable source data for the specified scripting component.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the [OSAID](#) (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in ["Mode Flags"](#) (page 102).

resultingScriptValueID

A pointer to the script ID for the script value returned.

Return Value

A result code. See ["Result Codes"](#) (page 114).

Discussion

The `OSACompileExecute` function compiles source data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, `OSACompileExecute` disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant `kOSANullScript`.

If the result code returned by `OSACompileExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACopyDisplayString

Converts a script value to an attributed Unicode text string, which your application can display to the user.

```
OSAEError OSACopyDisplayString (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    SInt32 modeFlags,
    CFAttributedStringRef *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script value to display. See the [OSAID](#) (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To make the resulting text readable by humans only, so that it can't be recompiled, specify `kOSAModeDisplayForHumans`.

result

If successful, a reference to the script data as an attributed Unicode text string; otherwise not defined. Because the `result` parameter returns a copy, the caller is responsible for releasing this string object, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSACopyDisplayString` function is analogous to [OSADisplay](#) (page 41), except that it returns the script text as an attributed Unicode text string. An instance of `CFAttributedString` manages a character string and an associated set of attributes that apply to characters or ranges of characters in the string. You can call [ASCopySourceAttributes](#) (page 17) to get the current AppleScript source style attributes.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`OSA.h`

OSACopyID

Updates script data after editing or recording and to perform undo or revert operations on script data.

```

OSAEError OSACopyID (
    ComponentInstance scriptingComponent,
    OSAID fromID,
    OSAID *toID
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

fromID

The script ID for script data that you want to be associated with the script ID in the *toID* parameter. See the [OSAID](#) (page 84) data type.

toID

A pointer to the script ID for the script data to be replaced. If the value of this parameter is `kOSANullScript`, the `OSACopyID` function returns a new script ID. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSACopyID` function replaces the script data identified by the script ID in the *toID* parameter with the script data identified by the script ID in the *fromID* parameter.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSACopyScriptingDefinition

Creates a copy of a scripting definition (sdef) from the specified file or bundle.

```

OSAEError OSACopyScriptingDefinition (
    const FSRef *ref,
    SInt32 modeFlags,
    CFDataRef *sdef
);

```

Parameters

ref

A file reference to the application file or bundle from which to copy the scripting definition.

modeFlags

Reserved for future use. Set to `kOSAModeNull`.

sdef

On return, the resulting scripting definition, as XML data.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

If the target application does not have a true scripting definition (sdef) but does have an 'aete' resource or a Cocoa script suite, this function translates the existing information to an sdef. As a result, `OSACopyScriptingDefinition` works for any scriptable application.

To provide a scripting definition in your application:

1. Put the sdef file in the `Resources` folder of the application bundle.
2. Add an entry to your information property list (`Info.plist`) file:
 - key: “OSAScriptingDefinition”
 - value: “MyApplication.sdef” (the name of the sdef)

For an introduction to scripting definitions, see “Specifying Scripting Terminology” in [AppleScript Overview](#). See the man page for `sdef(5)` for details of the sdef format.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`ASDebugging.h`

OSACopySourceString

Decompiles the script data for the specified script and returns a copy of the equivalent source data as an attributed Unicode text string.

```
OSAEError OSACopySourceString (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    SInt32 modeFlags,
    CFAttributedStringRef *result
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data to decompile. If you pass `kOSANullScript` in this parameter, `OSACopySourceString` returns a null source description (such as an empty text string). See the [OSAID](#) (page 84) data type.

modeFlags

No mode information is currently supported, so you should specify `kOSAModeNull` for this parameter.

result

If successful, a reference to the script data as an attributed Unicode text string; otherwise not defined.

Because the `result` parameter returns a copy, the caller is responsible for releasing this string object, according to the rules described in Ownership Policy in *Memory Management Programming Guide for Core Foundation*.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSACopySourceString` function is analogous to `OSAGetSource` (page 59), except that it returns the decompiled script data as an attributed Unicode text string (a Core Foundation attributed string object). This data can be displayed to the user or compiled and executed. You can call `ASCopySourceAttributes` (page 17) to get the current AppleScript source style attributes.

Availability

Available in Mac OS X v10.5 and later.

Declared In

OSA.h

OSADebuggerCreateSession

Do not use.

Unsupported

```
OSAEError OSADebuggerCreateSession (
    ComponentInstance scriptingComponent,
    OSAID inScript,
    OSAID inContext,
    OSADebugSessionRef *outSession
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerDisposeCallFrame

Do not use.

Unsupported

```
OSAEError OSADebuggerDisposeCallFrame (
    ComponentInstance scriptingComponent,
    OSADebugCallFrameRef inCallFrame
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerDisposeSession

Do not use.

Unsupported

```
OSAEError OSADebuggerDisposeSession (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetBreakpoint

Do not use.

Unsupported

```
OSAEError OSADebuggerGetBreakpoint (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    UInt32 inSrcOffset,
    OSAID *outBreakpoint
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetCallFrameState

Do not use.

Unsupported

```
OSAEError OSADebuggerGetCallFrameState (
    ComponentInstance scriptingComponent,
    OSADebugCallFrameRef inCallFrame,
    AERecord *outState
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetCurrentCallFrame

Do not use.

Unsupported

```
OSAEError OSADebuggerGetCurrentCallFrame (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    OSADebugCallFrameRef *outCallFrame
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetDefaultBreakpoint

Do not use.

Unsupported

```
OSAError OSADebuggerGetDefaultBreakpoint (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    OSAID *outBreakpoint
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetPreviousCallFrame

Do not use.

Unsupported

```
OSAError OSADebuggerGetPreviousCallFrame (
    ComponentInstance scriptingComponent,
    OSADebugCallFrameRef inCurrentFrame,
    OSADebugCallFrameRef *outPrevFrame
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetSessionState

Do not use.

Unsupported

```
OSAError OSADebuggerGetSessionState (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    AERecord *outState
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetStatementRanges

Do not use.

Unsupported

```
OSAEError OSADebuggerGetStatementRanges (  
    ComponentInstance scriptingComponent,  
    OSADebugSessionRef inSession,  
    AEDescList *outStatementRangeArray  
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerGetVariable

Do not use.

Unsupported

```
OSAEError OSADebuggerGetVariable (  
    ComponentInstance scriptingComponent,  
    OSADebugCallFrameRef inCallFrame,  
    const AEDesc *inVariableName,  
    OSAID *outVariable  
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerSessionStep

Do not use.

Unsupported

```
OSAEError OSADebuggerSessionStep (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    OSADebugStepKind inKind
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerSetBreakpoint

Do not use.

Unsupported

```
OSAEError OSADebuggerSetBreakpoint (
    ComponentInstance scriptingComponent,
    OSADebugSessionRef inSession,
    UInt32 inSrcOffset,
    OSAID inBreakpoint
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebuggerSetVariable

Do not use.

Unsupported

```
OSAEError OSADebuggerSetVariable (
    ComponentInstance scriptingComponent,
    OSADebugCallFrameRef inCallFrame,
    const AEDesc *inVariableName,
    OSAID inVariable
);
```

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADisplay

Converts a script value to text. Your application can then use its own functions to display this text to the user.

```
OSAEError OSADisplay (
    ComponentInstance scriptingComponent,
    OSAID scriptValueID,
    DescType desiredType,
    SInt32 modeFlags,
    AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptValueID

The script ID for the script value to coerce. See the [OSAID](#) (page 84) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To make the resulting text readable by humans only, so that it can't be recompiled, specify `kOSAModeDisplayForHumans`.

resultingText

A pointer to the resulting descriptor record.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSADisplay` function coerces the script value identified by *scriptValueID* to a descriptor record of the text type specified by the *desiredType* parameter, if possible. Valid types include the standard text descriptor types, plus any special types supported by the scripting component.

Unlike [OSAGetSource](#) (page 59), `OSADisplay` can coerce only script values and always produces a descriptor record of a text descriptor type. In addition, if you specify the mode flag `kOSAModeDisplayForHumans`, the resulting text cannot be recompiled.

If you want to get a script value in a form that you can display for humans to read, use `OSADisplay`. If you want the descriptor type of the descriptor record returned in the *resultingText* parameter to be the same as the descriptor type returned by a scripting component, use `OSACoerceToDesc` and specify `typeWildcard` as the desired type.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSADispose

Reclaims the memory occupied by script data.

```

OSAEError OSADispose (
    ComponentInstance scriptingComponent,
    OSAID scriptID
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data to be disposed of. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSADispose` function releases the memory assigned to the script data identified by the *scriptID* parameter. The script ID passed to the `OSADispose` function is no longer valid if the function returns successfully. A scripting component can then reuse that script ID for other script data.

A call to `OSADispose` returns `noErr` if the script ID is `kOSANullScript`, although it does not dispose of anything.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSADoEvent

Handles an Apple event with the aid of a script context and obtains a reply event.

```

OSAEError OSADoEvent (
    ComponentInstance scriptingComponent,
    const AppleEvent *theAppleEvent,
    OSAID contextID,
    SInt32 modeFlags,
    AppleEvent *reply
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

theAppleEvent

A pointer to the Apple event to be handled.

contextID

The script ID for the script context to be used to handle the Apple event. See the [OSAID](#) (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in “Mode Flags” (page 102).

reply

A pointer to the reply Apple event.

Return Value

A result code. See “Result Codes” (page 114).

Discussion

The `OSADoEvent` function resembles both `OSADoScript` and `OSAExecuteEvent`. However, unlike `OSADoScript`, the script `OSADoEvent` executes must be in the form of a script context, and execution is initiated by an Apple event. Unlike `OSAExecuteEvent`, `OSADoEvent` returns a reply Apple event rather than the script ID of the resulting script value.

The `OSADoEvent` function, like `OSAExecuteEvent`, attempts to use the script context specified by the *contextID* parameter to handle the Apple event specified by the *theAppleEvent* parameter. If the scripting component determines that the script context can't handle the event (for example, if a script written in an AppleScript dialect doesn't include statements that handle the event), `OSADoEvent` immediately returns `errAEEEventNotHandled` rather than `errOSAScriptError`. This causes the Apple Event Manager to look for an appropriate handler in the application's Apple event dispatch table or elsewhere, using standard Apple event dispatching.

If the scripting component determines that the script context can handle the event, `OSADoEvent` executes the script context's handler for the event and returns the resulting script ID.

The `OSADoEvent` function returns a reply event that contains either the resulting script value or, if an error occurred during script execution, information about the error. If the error `errOSAScriptError` occurs during script execution, `OSADoEvent` calls `OSAScriptError` and returns the appropriate error information in the reply. The `OSADoEvent` function never returns `errOSAScriptError`.

If the script context specifies that the Apple event should be passed to the application's standard handler for that event (for example, with an AppleScript `continue` statement), `OSADoEvent` passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event (that is, without calling `OSADoEvent` again). If the *contextID* parameter is `kOSANullScript`, the `OSADoEvent` function passes the event directly to the resume dispatch function. If the call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Special Considerations

Like `OSAExecuteEvent`, `OSADoEvent` can generate the result code `errAEEEventNotHandled` in at least two ways. If the scripting component determines that a script context doesn't declare a handler for a particular event, `OSADoEvent` immediately returns `errAEEEventNotHandled`. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, `OSADoEvent` returns `errAEEEventNotHandled` in the reply Apple event.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSADoScript

Compiles and executes a script and converts the resulting script value to text in a single step rather than calling `OSACompile`, `OSAExecute`, and `OSADisplay`.

```
OSAEError OSADoScript (
    ComponentInstance scriptingComponent,
    const AEDesc *sourceData,
    OSAID contextID,
    DescType desiredType,
    SInt32 modeFlags,
    AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceData

A pointer to a descriptor record identifying suitable source data for the specified scripting component.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the [OSAID](#) (page 84) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in ["Mode Flags"](#) (page 102).

resultingText

A pointer to the resulting descriptor record.

Return Value

A result code.

If the result code returned by `OSADoScript` is a general result code, there was some problem in arranging for the script to be run. If an error occurs during script execution, the error message of the error is stored in *resultingText*, and the function returns `errOSAScriptError`. You can use [OSAScriptError](#) (page 68) to obtain more information about the particular error.

For additional information on result codes, see ["Result Codes"](#) (page 114).

Discussion

Calling the `OSADoScript` function is equivalent to calling `OSACompile` followed by `OSAExecute` and `OSADisplay`. After compiling the source data, executing the compiled script using the script context identified by the *contextID* parameter, and returning the text equivalent of the resulting script value in the *resultingText* parameter, `OSADoScript` disposes of both the compiled script and the resulting script value.

Special Considerations

Prior to Mac OS X version 10.5, if an error occurred during script execution, the error message of the error was not returned in *resultingText*.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSADoScriptFile

Loads a script from the specified file, compiles the script if the file is a text file, executes the script, converts the resulting script value to text, and stores the script back into the file if the script has persistent properties and the file is not a text file.

```
OSAEError OSADoScriptFile (
    ComponentInstance scriptingComponent,
    const FSRef *scriptFile,
    OSAID contextID,
    DescType desiredType,
    SInt32 modeFlags,
    AEDesc *resultingText
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from and to save the script back to (if the script has persistent properties and the file is not a text file). See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSADoScriptFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is assumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 84) data type.

desiredType

The desired text descriptor type, such as `typeChar`, for the resulting descriptor record.

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompile` (page 30) (if the file is a text file), `OSADisplay` (page 41), `OSAExecute` (page 46), and `OSALoad` (page 62) functions. For descriptions of the mode flag usage of those functions, see the chapter “Scripting Components” in “Interapplication Communication” at <http://developer.apple.com/documentation/mac/IAC/IAC-2.html>.

resultingText

The descriptor record for the resulting script value. The `AEDesc` data type is described in Apple Event Manager Reference.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

This routine is effectively equivalent to calling `OSALoadFile` (page 65), followed by `OSAExecute` (page 46), `OSADisplay` (page 41), and then `OSAStoreFile` (page 80) if the script has persistent properties. After execution, the compiled source and the resulting value are disposed. Only the `resultingText` descriptor is retained. If an error occurs during script execution, the error message of the error is stored in `resultingText`, and the function returns `errOSAScriptError`. You can use `OSAScriptError` (page 68) to obtain more information about the particular error.

Special Considerations

Prior to Mac OS X version 10.5, if an error occurred during script execution, the error message of the error was not returned in `resultingText`.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`OSA.h`

OSAExecute

Executes a compiled script or a script context.

```
OSAEError OSAExecute (
    ComponentInstance scriptingComponent,
    OSAID compiledScriptID,
    OSAID contextID,
    SInt32 modeFlags,
    OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

compiledScriptID

The script ID for the compiled script to be executed. See the `OSAID` (page 84) data type.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in the description that follows.

resultingScriptValueID

A pointer to the script ID for the script value returned. See the `OSAID` (page 84) data type.

Return Value

A result code. See “[Result Codes](#)” (page 114). If the result code returned by `OSAExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Discussion

The `OSAExecute` function executes the compiled script identified by the `compiledScriptID` parameter, using the script context identified by the `contextID` parameter to maintain state information, such as the binding of variables, for the compiled script. After successfully executing a script, `OSAExecute` returns the script ID for a resulting script value, or, if execution does not result in a value, the constant `kOSANullScript`. You can use the `OSACoerceToDesc` function to coerce the resulting script value to a descriptor record of a desired descriptor type, or the `OSADisplay` (page 41) function to obtain the equivalent source data for the script value. You can control the way in which the scripting component executes a script by adding any of the flags described in “[Mode Flags](#)” (page 102).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAExecuteEvent

Handles an Apple event with the aid of a script context and obtains a script ID for the resulting script value.

```
OSAEError OSAExecuteEvent (
    ComponentInstance scriptingComponent,
    const AppleEvent *theAppleEvent,
    OSAID contextID,
    SInt32 modeFlags,
    OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

theAppleEvent

A pointer to the Apple event to be handled.

contextID

The script ID for the script context to be used to handle the Apple event. See the `OSAID` (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in “[Mode Flags](#)” (page 102).

resultingScriptValueID

A pointer to the script ID for the resulting script value.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

The `OSAExecuteEvent` function attempts to use the script context specified by the `contextID` parameter to handle the Apple event specified by the `theAppleEvent` parameter. If the scripting component determines that the script context can't handle the event (for example, if a script written in AppleScript doesn't include statements that handle the event), `OSAExecuteEvent` immediately returns `errAEventNotHandled` rather than `errOSAScriptError`. This causes the Apple Event Manager to look for an appropriate handler in the application's Apple event dispatch table or elsewhere, using standard Apple event dispatching.

If the scripting component determines that the script context can handle the event, `OSAExecuteEvent` executes the script context's handler and returns the resulting script ID. If execution of the script context's handler for the event generates an error, `OSAExecuteEvent` returns `errOSAScriptError`, and you can get more detailed error information by calling the `OSAScriptError` function.

If the script context identified by the `contextID` parameter specifies that the Apple event should be passed to the application's default handler for that event (for example, with an AppleScript `continue` statement), `OSAExecuteEvent` passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function dispatches the event directly to the application's standard handler for that event (that is, without calling `OSAExecuteEvent` again). If the `contextID` parameter is `kOSANullScript`, the `OSAExecuteEvent` function passes the event directly to the resume dispatch function. If a call to the resume dispatch function is successful, execution of the script context proceeds from the point at which the resume dispatch function was called.

Special Considerations

The `OSAExecuteEvent` function can generate the result code `errAEventNotHandled` in at least two ways. If the scripting component determines that a script context doesn't declare a handler for a particular event, `OSAExecuteEvent` immediately returns `errAEventNotHandled`. If a scripting component calls its resume dispatch function during script execution and the application's standard handler for the event fails to handle it, `OSAExecuteEvent` returns `errOSAScriptError` and a call to `OSAScriptError` with `kOSAErrorNumber` in the `selector` parameter returns `errAEventNotHandled` as the resulting error description.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAGenericToRealID

Converts a generic script ID to the corresponding component-specific script ID.

```
OSAEError OSAGenericToRealID (
    ComponentInstance genericScriptingComponent,
    OSAID *theScriptID,
    ComponentInstance *theExactComponent
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

theScriptID

A pointer to the generic script ID that you want to convert. The `OSAGenericToRealID` function returns, in this parameter, the component-specific script ID that corresponds to the generic script ID that you pass in this parameter. See the [OSAID](#) (page 84) data type.

theExactComponent

On return, a pointer to the component instance that created the script ID returned in the *theScriptID* parameter.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

You can't use the generic scripting component and a generic script ID with component-specific routines. Instead, you can use the component instance and script ID returned by `OSAGenericToRealID`.

Given a generic script ID (that is, a script ID returned by a call to a standard component routine via the generic scripting component), the `OSAGenericToRealID` function returns the equivalent component-specific script ID and the component instance that created that script ID. The `OSAGenericToRealID` function modifies the script ID in place, changing the generic script ID you pass in the *theScriptID* parameter to the corresponding component-specific script ID.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSAGetActiveProc

Gets a pointer to the active function that a scripting component is currently using.

```
OSALong OSAGetActiveProc (
    ComponentInstance scriptingComponent,
    OSALong *activeProc,
    SRefCon *refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

activeProc

On return, a pointer a UPP to the active function currently set for the specified scripting component.

refCon

On return, a pointer to the reference constant associated with the active function for the specified scripting component.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAGetCreateProc

Gets a pointer to the create function that a scripting component is currently using to create Apple events.

```
OSAEError OSAGetCreateProc (
    ComponentInstance scriptingComponent,
    OSACreateAppleEventUPP *createProc,
    SRefCon *refCon
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

createProc

On return, a pointer to the UPP to the create function currently set for the specified scripting component.

refCon

On return, a pointer to the reference constant associated with the create function for the specified scripting component.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAGetCurrentDialect

Gets the dialect code for the dialect currently being used by a scripting component.

```
OSAEError OSAGetCurrentDialect (
    ComponentInstance scriptingComponent,
    short *resultingDialectCode
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingDialectCode

On return, a pointer to the code for the current dialect of the specified scripting component.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAGetDefaultScriptingComponent

Gets the subtype code for the default scripting component associated with an instance of the generic scripting component.

```
OSAEError OSAGetDefaultScriptingComponent (
    ComponentInstance genericScriptingComponent,
    ScriptingComponentSelector *scriptingSubType
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptingSubType

On return, a pointer to the subtype code for the default scripting component associated with the instance of the generic scripting component specified in the *genericScriptingComponent* parameter.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAGetDefaultScriptingComponent` function returns the subtype code for the default scripting component. This is the scripting component that will be used by `OSASStartRecording`, `OSACompile`, or `OSACompileExecute` if no existing script ID is specified. From the user’s point of view, the default scripting component corresponds to the scripting language selected in the Script Editor application when the user first creates a new script.

Each instance of the generic scripting component has its own default scripting component, which is initially AppleScript. You can use `OSASetDefaultScriptingComponent` to change the default scripting component.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSAGeneric.h

OSAGetDialectInfo

Gets information about a specified dialect provided by a specified scripting component.

```

OSAEError OSAGetDialectInfo (
    ComponentInstance scriptingComponent,
    short dialectCode,
    OSType selector,
    AEDesc *resultingDialectInfo
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

dialectCode

A code for the dialect about which you want information. You can obtain a list of a scripting component's dialect codes by calling `OSAAvailableDialectCodeList`.

selector

A constant that indicates what kind of information you want `OSAGetDialectInfo` to return in the result parameter. This constant determines the descriptor type for the descriptor record returned. See the description in [“Dialect Descriptor Constants”](#) (page 94) for a list of the standard constants you can specify in this parameter.

resultingDialectInfo

A pointer to a descriptor record containing the requested information. The descriptor record's descriptor type corresponds to the constant specified in the *selector* parameter.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

After you obtain a list of dialect codes by calling `OSAAvailableDialectCodeList`, you can pass any of those codes to `OSAGetDialectInfo` to get information about the corresponding dialect. The descriptor type of the descriptor record returned by `OSAGetDialectInfo` depends on the constant specified in the *selector* parameter. All scripting components support the [“Dialect Descriptor Constants”](#) (page 94) constants for this parameter. Individual scripting components may allow you to specify additional constants.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAGetHandler

Gets a script ID for the specified script handler from the specified script.

```

OSAEError OSAGetHandler (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    const AEDesc *handlerName,
    OSAID *resultingCompiledScriptID
);

```

Parameters*scriptingComponent*

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

Specifies the script to get the script handler for. See the [OSAID](#) (page 84) data type.

handlerName

A descriptor record that specifies the name of the handler to get. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. The handler name is case-sensitive and must exactly match the case of the handler name as supplied by the `OSAGetHandlerNames` function or the `OSAGetSource` (page 59) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

resultingCompiledScriptID

On return, the `OSAID` for the specified handler, or `kOSANullScript` if the handler does not exist. If the handler has no input parameters, it may be executed by calling `OSAExecute`; if it requires input parameters, you can create an Apple event that supplies the handler parameters and execute it with `OSAExecuteEvent`. You may also copy it to another script with the `OSASetHandler` function or get its source code with the `OSAGetSource` function. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSAGetHandlerNames

Gets a list of all handler names in the specified script as an `AEDescList` of descriptors of type `typeChar`.

```

OSAEError OSAGetHandlerNames (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    AEDescList *resultingHandlerNames
);

```

Parameters*scriptingComponent*

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

See the [OSAID](#) (page 84) data type.

resultingHandlerNames

On return, a list of all handler names, as an `AEDescList` of descriptors of type `typeChar`. See [Apple Event Manager Reference](#) for a description of the `AEDescList` data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSAGetProperty

Gets the value of a specified script property from a specified script.

```

OSAEError OSAGetProperty (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    const AEDesc *variableName,
    OSAID *resultingScriptValueID
);

```

Parameters*scriptingComponent*

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

Specifies the script to get the script property from. See the [OSAID](#) (page 84) data type.

variableName

A descriptor record that specifies the name of the property to get. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. The variable name is case-sensitive and must exactly match the case of the variable name as supplied by the `OSAGetPropertyNames` function or the `OSAGetSource` (page 59) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

resultingScriptValueID

On return, a script ID whose associated data supplies the value for the property specified by the `variableName` parameter. Note that the value is returned as an `OSAID`, not an `AEDesc`. To get it as an `AEDesc`, use the `OSACoerceToDesc` function; to get it as user-readable text, use `OSADisplay` (page 41). See the `OSAID` (page 84) data type.

Return Value

A result code. See “Result Codes” (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSAGetPropertyNames

Gets a list of all property names from the specified script.

```
OSAEError OSAGetPropertyNames (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    AEDescList *resultingPropertyNames
);
```

Parameters*scriptingComponent*

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

contextID

Specifies the script to get the property names from. See the `OSAID` (page 84) data type.

resultingPropertyNames

On return, a list of all property names, as an `AEDescList` of descriptors of type `typeChar`. You can extract these descriptors from the list and use them as input values to the `OSAGetProperty` function or the `OSASetProperty` (page 73) function. See Apple Event Manager Reference for a description of the `AEDescList` data type.

Return Value

A result code. See “Result Codes” (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

ASDebugging.h

OSAGetResumeDispatchProc

Gets the resume dispatch function currently being used by a scripting component instance during execution of an AppleScript `continue` statement or its equivalent

```
OSAEError OSAGetResumeDispatchProc (
    ComponentInstance scriptingComponent,
    AEEEventHandlerUPP *resumeDispatchProc,
    SRefCon *refCon
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resumeDispatchProc

On return, a pointer to a UPP to the resume dispatch function for the specified scripting component. If no resume dispatch function has been registered, `OSAGetResumeDispatchProc` returns `kOSAUseStandardDispatch` (the default).

refCon

On return, a pointer to the reference constant associated with the resume dispatch function.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAGetScriptInfo

Obtains information about script data according to the value you pass in the `selector` parameter.

```
OSAEError OSAGetScriptInfo (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    OSType selector,
    long *result
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data about which to obtain information. See the [OSAID](#) (page 84) data type.

selector

A value that determines what kind of information `OSAGetScriptInfo` returns. The value can be one of the constants described in “[Script Information Selectors](#)” (page 109). In addition to the standard constants, the AppleScript component also supports the `kASHasOpenHandler` constant. For additional information, see the Version Notes section below.

result

On return, a pointer to the requested information, which you can coerce to the appropriate descriptor type for the value specified in the *selector* parameter.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Version Notes

In Mac OS X, if you specify `kOSAScriptIsModified` for the value of the *selector* parameter, `OSAGetScriptInfo` returns `true` if the script has been modified and `false` if it has not.

The following information describes the behavior of `OSAGetScriptInfo` in versions of the Mac OS prior to Mac OS X: Although you can specify `kOSAScriptIsModified` when you are using the AppleScript component without generating an error, the current version of AppleScript interprets this request conservatively. The AppleScript component stores script data in a network of interlocking structures, and running a script can cause any of these structures to be modified. If you pass a script ID to `OSAGetScriptInfo` with `kOSAScriptIsModified` as the value of the *selector* parameter, the AppleScript component returns 1 if there is any possibility that the script data or related structures may have been modified, and 0 if there is no possibility that they have been modified.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAGetScriptingComponent

Gets the instance of a scripting component for a specified subtype.

```
OSAEError OSAGetScriptingComponent (
    ComponentInstance genericScriptingComponent,
    ScriptingComponentSelector scriptingSubType,
    ComponentInstance *scriptingInstance
);
```

Parameters*genericScriptingComponent*

A component instance for the generic scripting component, created by a prior call to the `Component Manager` function `OpenDefaultComponent` or `OpenComponent`.

scriptingSubType

A subtype code for a scripting component.

scriptingInstance

On return, a pointer to a component instance for the scripting component identified by the *scriptingSubType* parameter.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

You can't use the generic scripting component with component-specific routines. Instead, use an instance of the specific scripting component, which you can obtain with `OSAGetScriptingComponent`.

The `OSAGetScriptingComponent` function returns, in the *scriptingInstance* parameter, an instance of the scripting component identified by the *scriptingSubType* parameter. Each instance of the generic scripting component keeps track of a single instance of each component subtype, so `OSAGetScriptingComponent` always returns the same instance of a specified scripting component that the generic scripting component uses for standard scripting component routines.

For example, you can use `OSAGetScriptingComponent` to get the subtype code for the default scripting component (that is, the scripting component used by the generic scripting component for new scripts). You can then get an instance of the default scripting component by passing its subtype code to `OSAGetScriptingComponent`. Finally, you can pass that instance to `OSAScriptingComponentName` to obtain the default scripting component's name so you can display it to the user.

Similarly, you can pass `kAppleScriptSubType` in the *scriptingSubType* parameter to obtain an instance of the AppleScript component. This is necessary, for example, to call AppleScript-specific routines such as `ASGetSourceStyles` (which is deprecated in Mac OS X version 10.5 in favor of [ASCopySourceAttributes](#) (page 17)).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSAGetScriptingComponentFromStored

Gets the subtype code for a scripting component that created a storage descriptor record.

```
OSAEError OSAGetScriptingComponentFromStored (
    ComponentInstance genericScriptingComponent,
    const AEDesc *scriptData,
    ScriptingComponentSelector *scriptingSubType
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the `Component Manager` function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to either a generic storage descriptor record or a component-specific storage descriptor record.

scriptingSubType

On return, a pointer to a subtype code identifying the scripting component that created the descriptor record specified by the *scriptData* parameter.

Return Value

A result code. See ["Result Codes"](#) (page 114).

Discussion

The `OSAGetScriptingComponentFromStored` function returns, in the `scriptingSubType` parameter, the subtype code for the scripting component that created the script data specified by the `scriptData` parameter.

The generic scripting component automatically identifies the appropriate scripting component for you when you use it to call `OSALoad`. By calling `OSAGetScriptingComponentFromStored`, you can determine, without loading a script, which scripting component created the script data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSAGetSendProc

Gets a pointer to the send function that a scripting component is currently using.

```
OSAEError OSAGetSendProc (
    ComponentInstance scriptingComponent,
    OSASendUPP *sendProc,
    SRefCon *refCon
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sendProc

On return, a pointer to the UPP to the send function currently set for the specified scripting component.

refCon

On return, a pointer to the reference constant associated with the send function for the specified scripting component.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAGetSource

Decompiles the script data identified by a script ID and obtains the equivalent source data.

```

OSAEError OSAGetSource (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    DescType desiredType,
    AEDesc *resultingSourceData
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data to decompile. If you pass `kOSANullScript` in this parameter, `OSAGetSource` returns a null source description (such as an empty text string). See the [OSAID](#) (page 84) data type.

desiredType

The desired descriptor type of the resulting descriptor record, or `typeBest` if any type will do.

resultingSourceData

A pointer to the resulting descriptor record.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAGetSource` function decompiles the script data identified by the specified script ID and returns a descriptor record containing the equivalent source data. The source data returned need not be exactly the same as the source data originally passed to `OSACompile`—for example, white space and formatting might be different—but it should be a reasonable equivalent suitable for user viewing and editing.

The difference between `OSACoerceToDesc` and `OSAGetSource` is that `OSAGetSource` creates source data that can be displayed to a user or compiled and executed to generate an appropriate value, whereas `OSACoerceToDesc` actually returns the value. For example, if you call `OSAGetSource` and specify a string value, it returns the text surrounded by quotation marks (so that it can be properly compiled). If you call `OSACoerceToDesc` and specify a string value, it simply returns the text.

The main difference between `OSADisplay` and `OSAGetSource` is that `OSAGetSource` can coerce any form of script data using a variety of descriptor types, whereas `OSADisplay` can coerce only script values and always produces a descriptor record of a text descriptor type.

A scripting component that supports the `OSAGetSource` function has the `kOSASupportsGetSource` bit set in the `componentFlags` field of its component description record.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAGetStorageType

Retrieves the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

```
OSErr OSAGetStorageType (
    AEDataStorage scriptData,
    DescType *dscType
);
```

Parameters*scriptData*

A handle to the script data.

dscType

A pointer to the descriptor type specified in the script data trailer.

Return ValueA result code. See [“Result Codes”](#) (page 114).**Discussion**

The `OSAGetStorageType` function retrieves the scripting component subtype from the trailer. If no trailer can be found, `OSAGetStorageType` returns the error `errOSABadStorageType`.

Availability

Available in Mac OS X v10.0 and later.

Declared In`OSAComp.h`**OSAGetSysTerminology**

Gets one or more scripting terminology resources from the OSA system.

```
OSAEError OSAGetSysTerminology (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    short terminologyID,
    AEDesc *terminologyList
);
```

Parameters*scriptingComponent*Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.*modeFlags*Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.*terminologyID**terminologyList*

On return, one or more terminology resources from the OSA system. These include the built-in terminology for AppleScript as well as the standard suites, but not the terminology for installed scripting additions. The terminology may be returned as a single `AEDesc` of type `typeAEUT` or as a list of such descriptors. The internal format of the `typeAEUT` descriptor is the resource format described in `AEUserTermTypes.r`. See [Apple Event Manager Reference](#) [Apple Event Manager Reference](#) for a description of the `AEDesc` data type.

Return ValueA result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

ASDebugging.h

OSALoad

Loads script data.

```
OSAEError OSALoad (
    ComponentInstance scriptingComponent,
    const AEDesc *scriptData,
    SInt32 modeFlags,
    OSAID *resultingScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to the descriptor record containing the script data to be loaded.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To indicate that only the minimum script data required to run the script should be loaded, pass `kOSAModePreventGetSource` in this parameter.

resultingScriptID

On return, a pointer to the script ID for the compiled script. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSALoad` function loads script data and returns a script ID. The generic scripting component uses the descriptor record in the *scriptData* parameter to determine which scripting component should load the script. If the descriptor record is of type `typeOSAGenericStorage`, the generic scripting component uses the trailer at the end of the script data to identify the scripting component. If the descriptor record's type is the subtype value for another scripting component, the generic scripting component uses the descriptor type to identify the scripting component.

If you want the script ID returned by `OSALoad` to identify only the minimum script data required to run the script and you are sure that you won't need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the *modeFlags* parameter.

Scripting components other than the generic scripting component can load script data only if it has been saved in a descriptor record whose descriptor type matches the scripting component's subtype.

Script data may change after it has been loaded—for example, if your application allows the user to edit a script's source data. To test whether script data has been modified, pass its script ID to `OSAGetScriptInfo`. If it has changed, you can call `OSAStore` again to obtain a handle to the modified script data and save it.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSALoadExecute

Loads and executes a script in a single step rather than calling OSALoad and OSAExecute.

```
OSALoadExecute (
    ComponentInstance scriptingComponent,
    const AEDesc *scriptData,
    OSAID contextID,
    SInt32 modeFlags,
    OSAID *resultingScriptValueID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptData

A pointer to the descriptor record identifying the script data to be loaded and executed.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the [OSAID](#) (page 84) data type.

modeFlags

Information used by individual scripting components. To avoid setting mode flag values, specify `kOSAModeNull`. Other possible mode flags are listed in [“Mode Flags”](#) (page 102).

resultingScriptValueID

A pointer to the script ID for the script value returned. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSALoadExecute` function loads script data and executes the resulting compiled script, using the script context identified by the *contextID* parameter to maintain state information such as the binding of variables. After successfully executing the script, `OSALoadExecute` disposes of the compiled script and returns either the script ID for the resulting script value or, if execution does not result in a value, the constant `kOSANullScript`.

You can control the way in which the scripting component executes a script by adding any of the [“Mode Flags”](#) (page 102) flags to the *modeFlags* parameter.

If the result code returned by `OSALoadExecute` is a general result code, there was some problem in arranging for the script to be run. If the result code is `errOSAScriptError`, an error occurred during script execution. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSALoadExecuteFile

Loads a script from the specified file into the specified scripting component, compiles the script if the file is a text file, and executes the script.

```
OSALoadExecuteFile (
    ComponentInstance scriptingComponent,
    const FSRef *scriptFile,
    OSAID contextID,
    SInt32 modeFlags,
    OSAID *resultingScriptValueID
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from. See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSALoadExecuteFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is presumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

contextID

The script ID for the context to be used during script execution. The constant `kOSANullScript` in this parameter indicates that the scripting component should use its default context. See the `OSAID` (page 84) data type.

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompileExecute` (page 31) (if the file is a text file) and `OSALoadExecute` (page 63) functions. For descriptions of the mode flag usage of those functions, see the chapter “Scripting Components” in “Interapplication Communication” at <http://developer.apple.com/documentation/mac/IAC/IAC-2.html>.

resultingScriptValueID

The script ID for the resulting script value. See the `OSAID` (page 84) data type.

Return Value

A result code. See “Result Codes” (page 114).

Discussion

This routine is effectively equivalent to calling `OSALoadFile` (page 65) followed by `OSAExecute` (page 46). After execution, the compiled source is disposed. Only the resulting value ID is retained.

Availability

Available in Mac OS X v10.3 and later.

Declared In

OSA.h

OSALoadFile

Loads a script from the specified file into the specified scripting component, compiling the script if the file is a text file.

```
OSAEError OSALoadFile (
    ComponentInstance scriptingComponent,
    const FSRef *scriptFile,
    Boolean *storable,
    SInt32 modeFlags,
    OSAID *resultingScriptID
);
```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptFile

Identifies the file to load the script from. See the File Manager documentation for a description of the `FSRef` data type.

File format is determined by inspection. If the file is a text file, `OSALoadFile` uses the following steps to determine the text encoding:

- If a Unicode BOM is present, that determines the encoding—one of UTF-16BE, UTF-16LE, or UTF-8
- Otherwise, if the file is valid UTF-8, it is presumed to be UTF-8.
- Otherwise, it is assumed to be in the primary encoding.

storable

If *storable* is not NULL, on return it is set to indicate whether a compiled script can be stored into the script file using `OSAStoreFile` (page 80).

modeFlags

Information for use by the scripting component. Can include any of the mode flags that would normally be sent to the `OSACompile` (page 30) (if the file is a text file) and `OSALoad` (page 62) functions. For descriptions of the mode flag usage of those functions, see the chapter “Scripting Components” in “Interapplication Communication” at <http://developer.apple.com/documentation/mac/IAC/IAC-2.html>.

resultingScriptID

The returned script ID for the compiled script. See the `OSAID` (page 84) data type.

Return Value

A result code. See “Result Codes” (page 114).

Availability

Available in Mac OS X v10.3 and later.

Declared In

OSA.h

OSAMakeContext

Gets a script ID for a new script context.

```

OSAEError OSAMakeContext (
    ComponentInstance scriptingComponent,
    const AEDesc *contextName,
    OSAID parentContext,
    OSAID *resultingContextID
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

contextName

A pointer to the name of the new context. Some scripting components may use context names for semantic purposes. If the value of this parameter is `typeNull`, `OSAMakeContext` creates an unnamed context.

parentContext

The existing context from which the new context inherits bindings. If the value of this parameter is `kOSANullScript`, the new context does not inherit bindings from any other context.

resultingContextID

A pointer to the script ID for the resulting script context. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAMakeContext` function creates a new script context that you may pass to `OSAExecute` or `OSAExecuteEvent`. The new script context inherits the bindings of the script context specified in the *parentContext* parameter.

If you call `OSAMakeContext` using an instance of the generic scripting component, the generic scripting component uses the default scripting component to create the new script context.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSAResultToGenericID

Converts a component-specific script ID to the corresponding generic script ID.

```

OSAEError OSARealToGenericID (
    ComponentInstance genericScriptingComponent,
    OSAID *theScriptID,
    ComponentInstance theExactComponent
);

```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the `Component Manager` function `OpenDefaultComponent` or `OpenComponent`.

theScriptID

A pointer to the component-specific script ID that you want to convert. You must have obtained this script ID from the scripting component instance passed in the *theExactComponent* parameter. The `OSARealToGenericID` function returns, in this parameter, the generic script ID that corresponds to the component-specific script ID that you pass in this parameter. See the [OSAID](#) (page 84) data type.

theExactComponent

A scripting component instance returned by a generic scripting component routine.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSARealToGenericID` function performs the reverse of the task performed by `OSAGenericToRealID`. Given a component-specific script ID and an exact scripting component instance (that is, the component instance that created the component-specific script ID), the `OSARealToGenericID` function returns the corresponding generic script ID. The `OSARealToGenericID` function modifies the script ID in place, changing the component-specific script ID passed in the *theScriptID* parameter to the corresponding generic script ID.

You'll need to do this if you have obtained a component-specific script ID using an exact scripting component instance and you want to refer to the same script in calls that use an instance of the generic scripting component. You can't use a component-specific script ID with the generic scripting component.

The script ID you pass in the *theScriptID* parameter must be a component-specific script ID obtained from a scripting component instance known to the generic scripting component. You can obtain such an instance by calling either `OSAGetScriptingComponent` or `OSAGenericToRealID`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSARemoveStorageType

Removes a trailer from the script data in a generic storage descriptor record

```

OSAEError OSARemoveStorageType (
    AEDataStorage scriptData
);

```

Parameters

scriptData

A handle to the script data.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSARemoveStorageType` function removes an existing trailer (reducing the handle's size). If no trailer can be found, then the handle is not modified, and `noErr` is returned.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAComp.h`

OSAScriptError

Gets information about errors that occur during script execution.

```
OSAEError OSAScriptError (
    ComponentInstance scriptingComponent,
    OSType selector,
    DescType desiredType,
    AEDesc *resultingErrorDescription
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

selector

A value that determines what `OSAScriptError` returns. The value can be one of the constants described in [“OSAScriptError Selectors”](#) (page 106).

desiredType

The desired descriptor type of the resulting descriptor record. The description that follows explains how this is determined by the value passed in the *selector* parameter.

resultingErrorDescription

On return, a pointer to the resulting descriptor record.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Whenever the `OSAExecute` function returns the error `errOSAScriptError`, you can use the `OSAScriptError` function to get more specific information about the error from the scripting component that encountered it. (This information remains available only until the next call to the same scripting component.) The information returned by `OSAScriptError` depends on the value passed in the *selector* parameter, which also determines the descriptor type you should specify in the *desiredType* parameter.

Every scripting component should support calls to `OSAScriptError` that pass `kOSAErrorNumber`, `kOSAErrorMessage`, or `kOSAErrorPartialResult` in the *selector* parameter.

Some scripting components may also support calls that pass other values in the selector parameter, including `kOSAErrorRange`, which provides start and end positions delimiting the errant expression in the source data. If the value of the *selector* parameter is `kOSAErrorRange`, the value of *desiredType* must be `typeOSAErrorRange`.

If the value of the selector parameter is `kOSAErrorNumber`, scripting components may return, in the *resultingErrorDescription* parameter, one of the general error codes described in “Result Codes” (page 114).

If you call `OSAScriptError` using an instance of the generic scripting component, the generic scripting component uses the same instance of a scripting component that it used for the previous call.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAScriptingComponentName

Gets the name of a scripting component.

```
OSAEError OSAScriptingComponentName (
    ComponentInstance scriptingComponent,
    AEDesc *resultingScriptingComponentName
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingScriptingComponentName

On return, a pointer to the name of the scripting component; or, if the component is the generic scripting component, the name of the default scripting component.

Return Value

A result code. See “Result Codes” (page 114).

Discussion

The `OSAScriptingComponentName` function returns a descriptor record that you can coerce to a text descriptor type such as `typeChar`. This can be useful if you want to display the name of the scripting language in which the user should write a new script.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASetActiveProc

Sets the active function that a scripting component calls periodically while executing a script.

```

OSAEError OSASetActiveProc (
    ComponentInstance scriptingComponent,
    OSAActiveUPP activeProc,
    SRefCon refCon
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

activeProc

A pointer to the active function to set. If the value of this parameter is `NULL`, `OSASetActiveProc` sets the scripting component's default active function.

refCon

A reference constant to be associated with the active function. This parameter can be used for many purposes; for example, it could contain a handle to data used by the active function.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSASetActiveProc` function allows your application to set a pointer to the active function called periodically by the scripting component during script execution. To get time periodically during script execution for its own purposes, your application can substitute its own active function for use by the scripting component. If you do not specify an active function, the scripting component uses its default active function, which allows a user to cancel script execution.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSASetCreateProc

Specifies a create function that a scripting component should use instead of the Apple Event Manager's `AECreatAppleEvent` function when creating Apple events.

```

OSAEError OSASetCreateProc (
    ComponentInstance scriptingComponent,
    OSACreateAppleEventUPP createProc,
    SRefCon refCon
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

createProc

A universal procedure pointer to the create function to set.

refCon

A reference constant.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

To gain control over the creation and addressing of Apple events, your application can provide its own create function for use by scripting components. To set a new create function, call the `OSASetCreateProc` function; to get the current create function, call `OSAGetCreateProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASetCurrentDialect

Sets the current dialect for a scripting component.

```
OSAEError OSASetCurrentDialect (
    ComponentInstance scriptingComponent,
    short dialectCode
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

dialectCode

The code for the dialect to be set.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASetDefaultScriptingComponent

Sets the default scripting component associated with an instance of the generic scripting component.

```
OSAEError OSASetDefaultScriptingComponent (
    ComponentInstance genericScriptingComponent,
    ScriptingComponentSelector scriptingSubType
);
```

Parameters

genericScriptingComponent

A component instance for the generic scripting component, created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptingSubType

The subtype code for the scripting component you want to set as the default.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

The `OSASetDefaultScriptingComponent` function sets the default scripting component for the specified instance of the generic scripting component to the scripting component identified by the *scriptingSubType* parameter.

Each instance of the generic scripting component has its own default scripting component, which is initially AppleScript. You can use `OSAGetDefaultScriptingComponent` to get the current default scripting component for an instance of the generic scripting component.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSAGeneric.h`

OSASetDefaultTarget

Sets the default target application for Apple events.

```
OSAEError OSASetDefaultTarget (
    ComponentInstance scriptingComponent,
    const AEResourceDesc *target
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

target

The address of the application that is being made the default application. If you pass a null descriptor record in this parameter, the scripting component treats the current process as the default target.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

Scripting components that support manipulation of the create and send functions also support the `OSASetDefaultTarget` function. The `OSASetDefaultTarget` function establishes the default target application for Apple event sending and the default application from which the scripting component should obtain terminology information. For example, AppleScript statements that refer to the default application do not need to be enclosed in `tell/end tell` statements.

If your application doesn't call this function, or if you pass a null descriptor record in the target parameter, the scripting component treats the current process (that is, the application that calls `OSAExecute` or related functions) as the default target application.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASetHandler

Sets a specified script handler in the specified script to the supplied handler.

```
OSAEError OSASetHandler (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    const AEDesc *handlerName,
    OSAID compiledScriptID
);
```

Parameters*scriptingComponent*

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. Pass the value `kOSAModeDontDefine` to prevent a handler from being created if it doesn't already exist. Otherwise, pass `kOSAModeNull` to avoid setting mode flag values (no other flags are applicable for this function).

contextID

Specifies the script to set the script handler for. See [OSAID](#) (page 84) for a description of the `OSAID` data type.

handlerName

A descriptor record that specifies the handler to set. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. If the handler does not already exist, it is created, unless you pass the value `kOSAModeDontDefine` for the `modeFlags` parameter. The handler name is case-sensitive and must exactly match the case of the handler name as supplied by the `OSAGetHandlerNames` function or the `OSAGetSource` (page 59) function. See [Apple Event Manager Reference](#) for a description of the `AEDesc` data type.

compiledScriptID

The `OSAID` value to set the handler to, normally obtained by a previous call to `OSAGetHandler`. Any other value will return an error value of `errOSAInvalidID`. Note that a script compiled by `OSACompile` is not itself a handler. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See ["Result Codes"](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

ASDebugging.h

OSASetProperty

Sets the value of a script property in a specified script, creating the property if it does not already exist.

```

OSAEError OSASetProperty (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    OSAID contextID,
    const AEDesc *variableName,
    OSAID scriptValueID
);

```

Parameters*scriptingComponent*

See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. Pass the value `kOSAModeDontDefine` to prevent a property from being created if it doesn't already exist in the specified script. Otherwise, pass `kOSAModeNull` to avoid setting mode flag values (no other flags are applicable for this function).

contextID

Specifies the script to set the script property for. See the [OSAID](#) (page 84) data type.

variableName

A descriptor record that specifies the name of the property to set. The descriptor must be of type `typeChar`, or of a type that can be coerced to `typeChar`. The variable name is case-sensitive and must exactly match the case of the variable name as supplied by the `OSAGetPropertyNames` function or the [OSAGetSource](#) (page 59) function. See Apple Event Manager Reference for a description of the `AEDesc` data type.

scriptValueID

A script ID whose associated data should be used to set the value for the property specified by `variableName`. Note that the value is specified by an `OSAID`, not an `AEDesc`. You can set a property to a value returned from script execution (from the `OSAExecute` function), extracted from another property (with the `OSAGetProperty` function), or converted from an `AEDesc` (by the `OSACoerceFromDesc` function). See the [OSAID](#) (page 84) data type.

Return Value

A result code. See ["Result Codes"](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Declared In

`ASDebugging.h`

OSASetResumeDispatchProc

Sets the resume dispatch function called by a scripting component during execution of an AppleScript `continue` statement or its equivalent.

```

OSAEError OSASetResumeDispatchProc (
    ComponentInstance scriptingComponent,
    AEEEventHandlerUPP resumeDispatchProc,
    SRefCon refCon
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resumeDispatchProc

A UPP to the resume dispatch function. You can specify one of the following in this parameter:

- a pointer to a resume dispatch function
- the `kOSAUseStandardDispatch` constant, which causes the Apple Event Manager to dispatch the event using standard Apple event dispatching (the handler registered in the application with `AEInstallEventHandler` is used)
- the `kOSANoDispatch` constant, which tells the Apple Event Manager that the processing of the Apple event is complete and that no dispatching should occur

refCon

A reference constant. You can pass the constant `kOSADontUsePhac` in this parameter, as described in the Discussion section below.

Return Value

A result code. See “[Result Codes](#)” (page 114).

Discussion

The `OSASetResumeDispatchProc` function sets the resume dispatch function that the specified instance of a scripting component calls during execution of an AppleScript continue statement or its equivalent. The resume dispatch function should dispatch the event to the application’s standard handler for that event.

If you are using a general handler for preliminary processing of Apple events, and if you can rely on standard Apple event dispatching to dispatch the event correctly, you don’t need to provide a resume dispatch function. Instead, you can specify `kOSAUseStandardDispatch` as the value of the *resumeDispatchProc* parameter and the constant `kOSADontUsePhac` as the value of the *refCon* parameter. This causes the Apple Event Manager to use standard Apple event dispatching except that it bypasses your application’s special handler dispatch table and thus won’t call your predispatch Apple event handler recursively. (A predispatch handler is called immediately before the Apple Event Manager dispatches an event.)

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

OSASetScriptInfo

Sets information about script data according to the value you pass in the selector parameter.

```

OSAEError OSASetScriptInfo (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    OSType selector,
    long value
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data whose information is to be set. See the [OSAID](#) (page 84) data type.

selector

A value that determines which information `OSASetScriptInfo` sets.

The value can be one of the constants described in [“Script Information Selectors”](#) (page 109). For more information, see the Version Notes section below.

In Mac OS X, the AppleScript component does not set a value.

value

The value to set.

In Mac OS X, the AppleScript component does not set a value.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Version Notes

In Mac OS X, if you specify `kOSAScriptIsModified` for the value of the *selector* parameter, it is ignored, and no value is set.

The following information describes the behavior of `OSASetScriptInfo` in versions of the Mac OS prior to Mac OS X: The `OSASetScriptInfo` function sets script information according to the value you pass in the *selector* parameter. If you use the `kOSAScriptIsModified` constant, `OSASetScriptInfo` sets a value that indicates whether the script data has been modified since it was created or passed to `OSALoad`. Some scripting components may provide additional constants.

For related information, see the [OSAGetScriptInfo](#) (page 56) function.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASetSendProc

Specifies a send function that a scripting component should use instead of the Apple Event Manger’s `AESend` function when sending Apple events.

```

OSAEError OSASetSendProc (
    ComponentInstance scriptingComponent,
    OSASendUPP sendProc,
    SRefCon refCon
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sendProc

A universal procedure pointer (UPP) to the send function to set.

refCon

A reference constant.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The send function provided by your application can perform almost any action instead of or in addition to sending Apple events; for example, it can be used to facilitate concurrent script execution. To set a new send function, call the `OSASetSendProc` function; to get the current send function, call `OSAGetSendProc`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAStartRecording

Turns on Apple event recording and records subsequent Apple events in a compiled script.

```

OSAEError OSAStartRecording (
    ComponentInstance scriptingComponent,
    OSAID *compiledScriptToModifyID
);

```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

compiledScriptToModifyID

A pointer to the script ID for the compiled script in which to record. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAStartRecording` function turns on Apple event recording. Subsequent Apple events are recorded (that is, appended to any existing statements) in the compiled script specified by the *compiledScriptToModifyID* parameter. If the source data for the compiled script is currently displayed in a script editor’s window, the script editor’s handler for the Recorded Text event should display each new

statement in the window as it is recorded. Users should not be able to change a script that is open in a script editor window while it is being recorded into. Recording continues until a call to `OSAStopRecording` turns recording off.

To record into a new compiled script, pass the constant `kOSANullScript` in the *compiledScriptToModifyID* parameter. The scripting component should respond by creating a new compiled script and recording into that.

The generic scripting component uses its default scripting component to create and record into a new compiled script.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAStopRecording

Turns off Apple event recording.

```
OSAEError OSAStopRecording (
    ComponentInstance scriptingComponent,
    OSAID compiledScriptID
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

compiledScriptID

A script ID for the compiled script into which Apple events are being recorded. See the [OSAID](#) (page 84) data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAStopRecording` function turns off recording. If the script is not currently open in a script editor window, the *compiledScriptToModifyID* parameter supplied to `OSAStartRecording` is then augmented to contain the newly recorded statements. If the script is currently open in a script editor window, the script data that corresponds to the *compiledScriptToModifyID* parameter supplied to `OSAStartRecording` is updated continuously until the client application calls `OSAStopRecording`.

If the compiled script identified by the script ID in the *compiledScriptID* parameter is not being recorded into or recording is not currently on, `OSAStopRecording` returns `noErr`.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAStore

Gets a handle to script data in the form of a storage descriptor record.

```
OSAError OSAStore (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    DescType desiredType,
    SInt32 modeFlags,
    AEDesc *resultingScriptData
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

scriptID

The script ID for the script data for which to obtain a data handle.

desiredType

The desired type of the descriptor record to be returned. If you want to store the script data in the form used by a generic storage descriptor record, specify `typeOSAGenericStorage`.

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To indicate that only the minimum script data required to run the script should be returned, pass `kOSAModePreventGetSource` in this parameter. (In this case the script data returned is not identical to the compiled script data and can't be used to generate source data.) If the `scriptID` parameter identifies a script context, you can pass `kOSAModeDontStoreParent` in this parameter to store the script context without storing its parent context.

resultingScriptData

On return, a pointer to the resulting descriptor record.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `OSAStore` function writes script data to a descriptor record so that the data can later be saved in a resource or written to the data fork of a document. You can then reload the data for the descriptor record as a compiled script (although possibly with a different script ID) by passing the descriptor record to `OSALoad`.

If you want the returned script data to be as small as possible and you are sure that you won't need to display the source data to the user, specify the `kOSAModePreventGetSource` flag in the `modeFlags` parameter. If the `scriptID` parameter identifies a script context and you don't want the returned script data to include the associated parent context, specify the `kOSAModeDontStoreParent` flag in the `modeFlags` parameter.

The desired type is either `typeOSAGenericStorage` (for a generic storage descriptor record) or a specific scripting component subtype value (for a component-specific storage descriptor record).

To store either a generic storage descriptor record or a component-specific storage descriptor record with your application's resources, use `'scpt'` as the resource type. The generic scripting component subtype, the generic storage descriptor type, and the resource type for stored script data all have the same value, though they serve different purposes.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSAStoreFile

Stores a script into the specified file.

```

OSAError OSAStoreFile (
    ComponentInstance scriptingComponent,
    OSAID scriptID,
    DescType desiredType,
    SInt32 modeFlags,
    const FSRef *scriptFile
);

```

Parameters*scriptingComponent*

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`. See the Component Manager documentation for a description of the `ComponentInstance` data type.

scriptID

Specifies the script to store. See the [OSAID](#) (page 84) data type.

desiredType

Specifies how the script should be stored. The desired type is either `typeOSAGenericStorage` (for a generic storage descriptor record) or a specific scripting component subtype value (for a component-specific storage descriptor record).

modeFlags

Information used by individual scripting components. To avoid setting any mode flags, specify `kOSAModeNull`. To indicate that only the minimum script data required to run the script should be stored, pass `kOSAModePreventGetSource` in this parameter. (In this case the stored script data is not identical to the compiled script data and can't be used to generate source data.) If the `scriptID` parameter identifies a script context, you can pass `kOSAModeDontStoreParent` in this parameter to store the script context without storing its parent context.

scriptFile

Identifies the file to store the script into. See the File Manager documentation for a description of the `FSRef` data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.3 and later.

Declared In

OSA.h

Callbacks

Your application can provide alternative active, send, and create functions for use by scripting components during script execution. All scripting components support routines that allow you to set and get the current active function called periodically by the scripting component during script execution. Some scripting components also support routines that allow you to set and get the current send and create functions used by the scripting component when it creates and sends Apple events during script execution.

OSAActiveProcPtr

Defines a pointer to an application-defined active function that performs periodic tasks during script compilation such as checking for Command-period, spinning the cursor, and checking for system-level errors.

```
typedef OSErr (*OSAActiveProcPtr) (
    long refCon
);
```

If you name your function `MyOSAActiveProc`, you would declare it like this:

```
OSErr MyOSAActiveProc (
    long refCon
);
```

Parameters

refCon

A reference constant.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Every scripting component calls an active function periodically during script compilation and execution and provides routines that allow your application to set or get the pointer to the active function.

If you don't set an alternative active function for a scripting component, it uses its own default active function. A scripting component's default active function allows a user to cancel script execution by pressing Command-period and calls `WaitNextEvent` to give other processes time.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACreateAppleEventProcPtr

Defines a pointer to an application-defined create function that allows you to gain control over the creation and addressing of Apple events.

```
typedef OSErr (*OSACreateAppleEventProcPtr)
(
    AEEEventClass theAEEEventClass,
    AEEEventID theAEEEventID,
    const AEAddressDesc * target,
    short returnID,
    long transactionID,
    AppleEvent * result,
    long refCon
);
```

If you name your function `MyOSACreateAppleEventProc`, you would declare it like this:

```
OSErr MyOSACreateAppleEventProc (
    AEEEventClass theAEEEventClass,
    AEEEventID theAEEEventID,
    const AEAddressDesc * target,
    short returnID,
    long transactionID,
    AppleEvent * result,
    long refCon
);
```

Parameters

theAEEEventClass

The event class of the Apple event to create.

theAEEEventID

The event ID of the Apple event to create.

target

A pointer to an address descriptor. This descriptor identifies the target (or server) application for the Apple event.

returnID

The return ID for the created Apple event.

transactionID

The transaction ID for this Apple event. A transaction is a series 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. The constant `kAnyTransactionID` specifies that the Apple event is not one of a series of interdependent Apple events.

result

A pointer to an Apple event. On successful return, this parameter should point to the new Apple event. On error, this should be a NULL descriptor.

refCon

A reference constant.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Every scripting component calls a create function whenever it creates an Apple event during script execution and provides routines that allow you to set or get the pointer to the create function.

Providing your own create function can be useful, for example, if your application needs to add its own transaction code to the event. An alternative create function takes the same parameters as the `AECreatAppleEvent` function plus a reference constant.

If you don't set an alternative create function for a scripting component, it uses the standard Apple Event Manager function `AECreatAppleEvent`, which it calls with its own default parameters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASendProcPtr

Defines a pointer to an application-defined send function that performs almost any action instead of or in addition to sending Apple events.

```
typedef OSErr (*OSASendProcPtr) (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    AESendMode sendMode,
    AESendPriority sendPriority,
    long timeOutInTicks,
    AEIdleUPP idleProc,
    AEFilerUPP filterProc,
    long refCon
);
```

If you name your function `MyOSASendProc`, you would declare it like this:

```
OSErr MyOSASendProc (
    const AppleEvent * theAppleEvent,
    AppleEvent * reply,
    AESendMode sendMode,
    AESendPriority sendPriority,
    long timeOutInTicks,
    AEIdleUPP idleProc,
    AEFilerUPP filterProc,
    long refCon
);
```

Parameters

theAppleEvent

A pointer to the Apple event.

reply

A pointer to a reply Apple event.

sendMode

Specifies various options for how the Apple event should be handled.

sendPriority

A value that specifies the priority for processing the Apple event.

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 before timing out. If this parameter is `kNoTimeout`, the Apple event never times out.

idleProc

A universal procedure pointer to a function that handles events (such as update, operating-system, activate, and null events) received while waiting for a reply.

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. This parameter may be `NULL`.

refCon

A reference constant.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

Every scripting component calls a send function whenever it sends an Apple event during script execution and provides routines that allow you to set or get the pointer to the send function.

For example, before sending an Apple event, an alternative send function can modify the event or save a copy of the event. An alternative send function takes the same parameters as the `AESend` function plus a reference constant.

If you don't set an alternative send function for a scripting component, it uses the standard Apple Event Manager function `AESend`, which it calls with its own default parameters.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`OSA.h`

Data Types

OSAID

Used by a scripting component to keep track of script data in memory.

```
typedef unsigned long OSAID;
```

Discussion

A scripting component assigns a script ID when it creates the associated script data (that is, a compiled script, a script value, a script context, or other kinds of script data supported by a scripting component) or loads it into memory. The scripting routines that create, load, compile, and execute scripts all return script IDs, and your application must pass valid script IDs to the other routines that manipulate scripts. A script ID remains valid until a client application calls `OSADispose` to reclaim the memory used for the corresponding script data.

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

GenericID

Represents the ID for generic scripting components.

```
typedef OSAID GenericID;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSAGeneric.h

OSAEError

Represents an OSA result code.

```
typedef ComponentResult OSAError;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

ScriptingComponentSelector

```
typedef OSType ScriptingComponentSelector;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSAGeneric.h

StatementRange

```
struct StatementRange {
    unsigned long startPos;
    unsigned long endPos;
};
typedef struct StatementRange StatementRange;
```

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSAActiveUPP

Defines a universal procedure pointer (UPP) to an application-defined active function.

```
typedef OSAActiveProcPtr OSAActiveUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSACreateAppleEventUPP

Defines a universal procedure pointer (UPP) to an application-defined Apple event creation function.

```
typedef OSACreateAppleEventProcPtr OSACreateAppleEventUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSASendUPP

Defines a universal procedure pointer (UPP) to an application-defined send function.

```
typedef OSASendProcPtr OSASendUPP;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

OSA.h

OSADebugCallFrameRef

```
typedef OSAID OSADebugCallFrameRef;
```

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

OSADebugSessionRef

```
typedef OSAID OSADebugSessionRef;
```

Availability

Available in Mac OS X v10.0 through Mac OS X v10.4.

Declared In

OSA.h

Constants

cClosure

```
enum {
    cClosure = 'clsr',
    cRawData = 'rdat',
    cStringClass = typeChar,
    cNumber = 'nubr',
    cListElement = 'celm',
    cListOrRecord = 'lr ',
    cListOrString = 'ls ',
    cListRecordOrString = 'lrs ',
    cNumberOrString = 'ns ',
    cNumberOrDateTime = 'nd ',
    cNumberDateTimeOrString = 'nds ',
    cAliasOrString = 'sf ',
    cSeconds = 'scnd',
    typeSound = 'snd ',
    enumBooleanValues = 'boov',
    kAETrue = typeTrue,
    kAEFalse = typeFalse,
    enumMiscValues = 'misc',
    kASCurentApplication = 'cura',
    formUserPropertyID = 'usrp'
};
```

cCoercion

```
enum {
    cCoercion = 'coec',
    cCoerceUpperCase = 'txup',
    cCoerceLowerCase = 'txlo',
    cCoerceRemoveDiacriticals = 'txdc',
    cCoerceRemovePunctuation = 'txpc',
    cCoerceRemoveHyphens = 'txhy',
    cCoerceOneByteToTwoByte = 'txex',
    cCoerceRemoveWhiteSpace = 'txws',
    cCoerceSmallKana = 'txsk',
    cCoerceZenkakuhankaku = 'txze',
    cCoerceKataHiragana = 'txkh',
    cZone = 'zone',
    cMachine = 'mach',
    cAddress = 'addr',
    cRunningAddress = 'radd',
    cStorage = 'stor'
};
```

cHandleBreakpoint

```
enum {
    cHandleBreakpoint = 'brak'
```



```
};
```

Component Flags

Indicate which features a scripting component supports.

```
enum {
    kOSASupportsCompiling = 0x0002,
    kOSASupportsGetSource = 0x0004,
    kOSASupportsAECOercion = 0x0008,
    kOSASupportsAESending = 0x0010,
    kOSASupportsRecording = 0x0020,
    kOSASupportsConvenience = 0x0040,
    kOSASupportsDialects = 0x0080,
    kOSASupportsEventHandling = 0x0100
};
```

Constants

`kOSASupportsCompiling`

Set if the scripting component supports the functions described in [“Compiling Scripts”](#) (page 10).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsGetSource`

Set if the scripting component supports the `OSAGetSource` function.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsAECOercion`

Set if the scripting component supports the `OSACoerceFromDesc` and `OSACoerceToDesc` functions.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsAESending`

Set if the scripting component supports the functions described in [“Manipulating the Create and Send Functions”](#) (page 11).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsRecording`

Set if the scripting component supports the `OSAStartRecording` (page 77) and `OSAStopRecording` (page 78) functions.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSASupportsConvenience`

Set if the script component supports the `OSALoadExecute` (page 63), `OSACompileExecute` (page 31), and `OSADoScript` (page 44) functions.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSASupportsDialects

Set if the scripting component supports the [OSASetCurrentDialect](#) (page 71), [OSAGetCurrentDialect](#) (page 50), [OSAAvailableDialectCodeList](#) (page 27), [OSAGetDialectInfo](#) (page 51), and [OSAAvailableDialects](#) (page 27) functions.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSASupportsEventHandling

Set if the scripting component supports the event handling functions described in [“Using Script Contexts to Handle Apple Events”](#) (page 13).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Discussion

Your application can use the Component Manager to find a scripting component that supports a specific group of functions or to determine whether a particular scripting component supports a specific group of functions. Each of these flags identifies one of these groups of functions. To specify one or more groups of functions for the Component Manager, use these constants to set the equivalent bits in the `componentFlags` field of a component description record.

Declared In

`OSA.h`

Considerations Flags

```
enum {
    kAECASE                    = 'case',
    kAEDIACRITIC               = 'diac',
    kAEWHITESPACE              = 'whit',
    kAEHYPHENS                  = 'hyph',
    kAEEXPANSION                = 'expa',
    kAEPUNCTUATION             = 'punc',
    kAEZENKAKUHANKAKU         = 'zkhk',
    kAESMALLKANA                = 'skna',
    kAEKATAHIRAGANA            = 'hika',
    kASCONSIDERREPLIES         = 'rmte',
    kASNUMERICSTRINGS          = 'nume',
    enumConsiderations         = 'cons'
};
```

Constants**kASNumericStrings**

Should strings be considered as numbers?

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

Version Notes

The constant `kASNumericStrings` is available starting with Mac OS X version 10.4.

Declared In

`ASRegistry.h`

Considerations Bit Masks

Specify settings for string comparisons.

```
enum {
    kAECASEConsiderMask           = 0x00000001,
    kAEDIACRITICConsiderMask     = 0x00000002,
    kAEWHITESPACEConsiderMask    = 0x00000004,
    kAEHYPHENSConsiderMask       = 0x00000008,
    kAEEXPANSIONConsiderMask     = 0x00000010,
    kAEPUNCTUATIONConsiderMask   = 0x00000020,
    kASCONSIDERREPLIESConsiderMask = 0x00000040,
    kASNUMERICSTRINGSConsiderMask = 0x00000080,
    kAECASEIgnoreMask            = 0x00010000,
    kAEDIACRITICIgnoreMask       = 0x00020000,
    kAEWHITESPACEIgnoreMask      = 0x00040000,
    kAEHYPHENSIgnoreMask         = 0x00080000,
    kAEEXPANSIONIgnoreMask       = 0x00100000,
    kAEPUNCTUATIONIgnoreMask     = 0x00200000,
    kASCONSIDERREPLIESIgnoreMask = 0x00400000,
    kASNUMERICSTRINGSIgnoreMask  = 0x00800000,
    enumConsidsAndIgnores        = 'csig'
};
```

Constants

`kASNumericStringsConsiderMask`

If bit at this position is set, consider strings to represent numerical values for comparison. For example, compare the string “1.01” as if it were the number 1.01.

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

`kASNumericStringsIgnoreMask`

If bit at this position is set, do not compare strings as numeric values.

Available in Mac OS X v10.4 and later.

Declared in `ASRegistry.h`.

Discussion

AppleScript has various settings for string comparisons, such as whether to consider or ignore capitalization. When your application receives an Apple event from AppleScript, it contains an attribute with the keyword `enumConsidsAndIgnores`. You can extract the consideration bit information from that attribute as `typeSInt32`, then use the bit masks in this enum to determine which considering and ignoring flags are currently set. You can use that information to conduct comparisons with the same criteria currently in use by AppleScript.

Version Notes

The constants `kASNumericStringsConsiderMask` and `kASNumericStringsIgnoreMask` are available starting with Mac OS X version 10.4.

Declared In

`ASRegistry.h`

cString

```
enum {
    cString = cStringClass
};
```

Current Dialect Constants

```
enum {
    kOSASelectSetCurrentDialect = 0x0701,
    kOSASelectGetCurrentDialect = 0x0702,
    kOSASelectAvailableDialects = 0x0703,
    kOSASelectGetDialectInfo = 0x0704,
    kOSASelectAvailableDialectCodeList = 0x0705
};
```

Discussion

AppleScript is designed so that scripts can be displayed in different dialects, which are representations of AppleScript that resemble human languages or programming languages. While dialects are supported, they are not particularly useful because no currently available OSA language supports dialects other than English.

Date and Time Constants

```
enum {
    pASWeekday = 'wkdy',
    pASMonth = 'mnth',
    pASDay = 'day ',
    pASYear = 'year',
    pASTime = 'time',
    pASDateString = 'dstr',
    pASTimeString = 'tstr',
    cMonth = pASMonth,
    cJanuary = 'jan ',
    cFebruary = 'feb ',
    cMarch = 'mar ',
    cApril = 'apr ',
    cMay = 'may ',
    cJune = 'jun ',
    cJuly = 'jul ',
    cAugust = 'aug ',
    cSeptember = 'sep ',
    cOctober = 'oct ',
    cNovember = 'nov ',
    cDecember = 'dec '
};
```

Default Initialization Values

Initialization constants passed to `ASInit` function.

```
enum {
    kASDefaultMinStackSize = 4 * 1024,
    kASDefaultPreferredStackSize = 16 * 1024,
    kASDefaultMaxStackSize = 16 * 1024,
    kASDefaultMinHeapSize = 4 * 1024,
    kASDefaultPreferredHeapSize = 16 * 1024,
    kASDefaultMaxHeapSize = 32L * 1024 * 1024
};
```

Constants**kASDefaultMinStackSize**

Represents the default value for the minimum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

kASDefaultPreferredStackSize

Represents the default value for the preferred size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

kASDefaultMaxStackSize

Represents the default value for the maximum size for the portion of the application's heap used by the AppleScript component's application-specific stack.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

kASDefaultMinHeapSize

Represents the default value for the minimum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

kASDefaultPreferredHeapSize

Represents the default value for the preferred size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

kASDefaultMaxHeapSize

Represents the default value for the maximum size for the portion of the application's heap used by the AppleScript component's application-specific heap. (See Version Notes section.)

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

Discussion

You can pass these constants to the `ASInit` (page 20) function to use the default values when initializing the AppleScript component. These values are also used if `ASInit` is not called explicitly, or if any of `ASInit`'s parameters are zero.

Version Notes

Starting in Mac OS X version 10.5, heap size parameter values are ignored—AppleScript's heap will grow as large as needed.

Declared In

AppleScript.h

Dialect Descriptor Constants

Define the descriptor type and keywords for descriptor records describing the dialects supported by a scripting component.

```
enum {
    typeOSADialectInfo = 'difo',
    keyOSADialectName = 'dnam',
    keyOSADialectCode = 'dcod',
    keyOSADialectLangCode = 'dlcd',
    keyOSADialectScriptCode = 'dscd'
};
```

Constants

typeOSADialectInfo

The descriptor type for each item in list returned by `OSAAvailableDialects`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

keyOSADialectName

Used with a descriptor record of any text type, such as type `typeChar`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

keyOSADialectCode

Used with a descriptor record of type `typeShortInteger`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

keyOSADialectLangCode

Used with a descriptor record of type `typeShortInteger`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

keyOSADialectScriptCode

Used with a descriptor record of type `typeShortInteger`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Discussion

These constants define the descriptor type for each item in the list returned by `OSAAvailableDialects` and the keywords for descriptor records of that type. The keyword constants can also be used in the *selector* parameter of `OSAGetDialectInfo` to obtain information about the dialects supported by a scripting component.

Generic Scripting Component Selectors

```
enum {
    kGSSSelectGetDefaultScriptingComponent = 0x1001,
    kGSSSelectSetDefaultScriptingComponent = 0x1002,
    kGSSSelectGetScriptingComponent = 0x1003,
    kGSSSelectGetScriptingComponentFromStored = 0x1004,
    kGSSSelectGenericToRealID = 0x1005,
    kGSSSelectRealToGenericID = 0x1006,
    kGSSSelectOutOfRange = 0x1007
};
```

Global Properties

```
enum {
    pASIt = 'it ',
    pASMe = 'me ',
    pASResult = 'rslt',
    pASSpace = 'spac',
    pASReturn = 'ret ',
    pASTab = 'tab ',
    pASPi = 'pi ',
    pASParent = 'pare',
    kASInitializeEventCode = 'init',
    pASPrintLength = 'prln',
    pASPrintDepth = 'prdp',
    pASTopLevelScript = 'ascr'
};
```

kASAdd

```
enum {
    kASAdd = '+ ',
    kASSubtract = '- ',
    kASMultiply = '* ',
    kASDivide = '/ ',
    kASQuotient = 'div ',
    kASRemainder = 'mod ',
    kASPower = '^ ',
    kASEqual = kAEEquals,
    kASNotEqual = ' ',
    kASGreaterThan = kAEGreaterThan,
    kASGreaterThanOrEqual = kAEGreaterThanEquals,
    kASLessThan = kAELessThan,
    kASLessThanOrEqual = kAELessThanEquals,
    kASComesBefore = 'cbfr',
    kASComesAfter = 'cafr',
    kASConcatenate = 'ccat',
    kASStartsWith = kAEBeginsWith,
    kASEndsWith = kAEEndsWith,
    kASContains = kAEContains
};
```

kASAnd

```
enum {
    kASAnd = kAEAND,
    kASOr = kAEOR,
    kASNot = kAENOT,
    kASNegate = 'neg ',
    keyASArg = 'arg '
};
```

kASErrorEventCode

```
enum {
    kASErrorEventCode = 'err ',
    kOSAErrorArgs = 'erra',
    keyAErrorObject = 'erob',
    pLength = 'leng',
    pReverse = 'rvse',
    pRest = 'rest',
    pInherits = 'c@#^',
    pProperties = 'pALL',
    keyASUserRecordFields = 'usrf',
    typeUserRecordFields = typeAEList
};
```

kASStartLogEvent

```
enum {
    kASStartLogEvent = 'log1',
    kASStopLogEvent = 'log0',
    kASCommentEvent = 'cmnt'
};
```

kDialectBundleResType

```
enum {
    kDialectBundleResType = 'Dbdl',
    cConstant = typeEnumerated,
    cClassIdentifier = pClass,
    cObjectBeingExamined = typeObjectBeingExamined,
    cList = typeAEList,
    cSmallReal = typeSMFloat,
    cReal = typeFloat,
    cRecord = typeAERecord,
    cReference = cObjectSpecifier,
    cUndefined = 'undf',
    cMissingValue = 'msng',
    cSymbol = 'symb',
    cLinkedList = 'llst',
    cVector = 'vect',
    cEventIdentifier = 'evnt',
    cKeyIdentifier = 'kyid',
    cUserIdentifier = 'uid ',
};
```



```

    cPreposition = 'prep',
    cKeyForm = enumKeyForm,
    cScript = 'scpt',
    cHandler = 'hand',
    cProcedure = 'proc'
};

```

keyAETarget

```

enum {
    keyAETarget = 'targ',
    keySubjectAttr = 'subj',
    keyASReturning = 'Krtn',
    kASAppleScriptSuite = 'ascr',
    kASScriptEditorSuite = 'ToyS',
    kASTypeNamesSuite = 'tpnm',
    typeAETE = 'aete',
    typeAEUT = 'aeut',
    kGetAETE = 'gdte',
    kGetAEUT = 'gdut',
    kUpdateAEUT = 'udut',
    kUpdateAETE = 'udte',
    kCleanupAEUT = 'cdut',
    kASComment = 'cmnt',
    kASLaunchEvent = 'noop',
    keyScszResource = 'scsz',
    typeScszResource = 'scsz',
    kASSubroutineEvent = 'psbr',
    keyASSubroutineName = 'snam',
    kASPrepositionalSubroutine = 'psbr',
    keyASPositionalArgs = 'parg'
};

```

keyAppHandledCoercion

```

enum {
    keyAppHandledCoercion = 'idas'
};

```

keyASPrepositionAt

```

enum {
    keyASPrepositionAt = 'at ',
    keyASPrepositionIn = 'in ',
    keyASPrepositionFrom = 'from',
    keyASPrepositionFor = 'for ',
    keyASPrepositionTo = 'to ',
    keyASPrepositionThru = 'thru',
    keyASPrepositionThrough = 'thgh',
    keyASPrepositionBy = 'by ',
    keyASPrepositionOn = 'on ',
    keyASPrepositionInto = 'into',
    keyASPrepositionOnto = 'onto',
};

```

```

keyASPrepositionBetween = 'btwn',
keyASPrepositionAgainst = 'agst',
keyASPrepositionOutOf = 'outo',
keyASPrepositionInsteadOf = 'isto',
keyASPrepositionAsideFrom = 'asdf',
keyASPrepositionAround = 'arnd',
keyASPrepositionBeside = 'bsid',
keyASPrepositionBeneath = 'bnth',
keyASPrepositionUnder = 'undr'
};

```

keyASPrepositionOver

```

enum {
    keyASPrepositionOver = 'over',
    keyASPrepositionAbove = 'abve',
    keyASPrepositionBelow = 'belw',
    keyASPrepositionApartFrom = 'aprt',
    keyASPrepositionGiven = 'givn',
    keyASPrepositionWith = 'with',
    keyASPrepositionWithout = 'wout',
    keyASPrepositionAbout = 'abou',
    keyASPrepositionSince = 'snce',
    keyASPrepositionUntil = 'till'
};

```

keyOSASourceEnd

Specifies the end of an error range.

```

enum {
    keyOSASourceEnd = 'srce'
};

```

Constants

keyOSASourceEnd

Field of a typeOSAErrorRange record of typeShortInteger. This field specifies the end of the error range.

Available in Mac OS X v10.0 and later.

Declared in OSA.h.

Declared In

OSA.h

keyOSASourceStart

Specifies the start of an error range.

```
enum {  
    keyOSASourceStart = 'srcs'  
};
```

Constants

`keyOSASourceStart`

Field of a `typeOSAErrorRange` record of `typeShortInteger`. This field specifies the start of the error range.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

keyProcedureName

```
enum {  
    keyProcedureName = 'dfnm',  
    keyStatementRange = 'dfsR',  
    keyLocalsNames = 'dfln',  
    keyGlobalsNames = 'dfgn',  
    keyParamsNames = 'dfpn'  
};
```

keyProgramState

```
enum {  
    keyProgramState = 'dsps'  
};
```

kGenericComponentVersion

Specifies the generic component version.

```
enum {  
    kGenericComponentVersion = 0x0100  
};
```

Constants

`kGenericComponentVersion`

Indicates the component version this header file describes.

Available in Mac OS X v10.0 and later.

Declared in `OSAGeneric.h`.

Declared In

`OSAGeneric.h`

kOSAComponentType

Defines the Component Manager type code for components that support the standard scripting component routines.

```
enum {
    kOSAComponentType = 'osa '
};
```

Constants

kOSAComponentType
 Specifies the standard OSA component type.
 Available in Mac OS X v10.0 and later.
 Declared in `OSA.h`.

Declared In

`OSA.h`

kOSAGenericScriptingComponentSubtype

Defines the subtype code for the generic scripting component.

```
enum {
    kOSAGenericScriptingComponentSubtype = 'scpt'
};
```

kOSAModeDontDefine

```
enum {
    kOSAModeDontDefine = 0x0001
};
```

Constants

kOSAModeDontDefine
 This mode flag can be passed to [OSASetProperty](#) (page 73) or [OSASetHandler](#) (page 73) and will prevent properties or handlers from being defined in a context that doesn't already have bindings for them. An error is returned if a current binding doesn't already exist.
 Available in Mac OS X v10.0 and later.
 Declared in `ASDebugging.h`.

kOSANullScript

Defines a null script ID.

```
enum {
    kOSANullScript = 0
};
```

Discussion

If the execution of a script does not result in a value, `OSAExecute` returns the constant `kOSANullScript` as the script ID. If a client application passes `kOSANullScript` to the `OSAGetSource` function instead of a valid script ID, the scripting component should display a null source description (possibly an empty text string). If a client application passes `kOSANullScript` to `OSAStartRecording`, the scripting component creates a new compiled script for editing or recording.

kOSARecordedText

Defines the event code for the Recorded Text event.

```
enum {
    kOSARecordedText = 'recd'
};
```

kOSAScriptResourceType

Defines the resource type for stored script data.

```
enum {
    kOSAScriptResourceType = kOSAGenericScriptingComponentSubtype
};
```

Constants

`kOSAScriptResourceType`
Resource type for scripts.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

kOSASelectComponentSpecificStart

```
enum {
    kOSASelectComponentSpecificStart = 0x1001
};
```

Constants

`kOSASelectComponentSpecificStart`
Scripting component specific selectors are added beginning with this value.
Available in Mac OS X v10.0 and later.
Declared in `OSA.h`.

kOSASelectCopyScript

```
enum {  
    kOSASelectCopyScript = 0x0105  
};
```

kOSASuite

Defines the suite for the Recorded Text event.

```
enum {  
    kOSASuite = 'ascr'  
};
```

Mode Flags

Specify information used by the scripting component.

```

enum {
    kOSAModePreventGetSource = 0x00000001
};
enum {
    kOSAModeNeverInteract = kAENeverInteract,
    kOSAModeCanInteract = kAECanInteract,
    kOSAModeAlwaysInteract = kAEAlwaysInteract,
    kOSAModeDontReconnect = kAEDontReconnect
};
enum {
    kOSAModeCantSwitchLayer = 0x00000040
};
enum {
    kOSAModeDoRecord = 0x00001000
};
enum {
    kOSAModeCompileIntoContext = 0x00000002
};
enum {
    kOSAModeAugmentContext = 0x00000004
};
enum {
    kOSAModeDisplayForHumans = 0x00000008
};
enum {
    kOSAModeDontStoreParent = 0x00010000
};
enum {
    kOSAModeDispatchToDirectObject = 0x00020000
};
enum {
    kOSAModeDontGetDataForArguments = 0x00040000
};
enum {
    kOSAModeFullyQualifyDescriptors = 0x00080000
};
};

```

Constants

`kOSAModePreventGetSource`

This mode flag may be passed to [OSALoad](#) (page 62), [OSAStore](#) (page 79), or [OSACompile](#) (page 30) to instruct the scripting component to not retain the “source” of an expression. This will cause a call to [OSAGetSource](#) (page 59) to return the error `errOSASourceNotAvailable` if used. However, some scripting components may not retain the source anyway. This is mainly used when either space efficiency is desired, or a script is to be “locked” so that its implementation may not be viewed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeNeverInteract`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether or not the script may interact with the user if necessary. Adds `kAENeverInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCanInteract`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether or not the script may interact with the user. Adds `kAECanInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeAlwaysInteract`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether or not the script may interact with the user. Adds `kAEAAlwaysInteract` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontReconnect`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether or not the script may reconnect if necessary. Adds `kAEDontReconnect` to the `sendMode` parameter of `AESend` for events sent when the script is executed.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCantSwitchLayer`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether Apple events should be sent with the `kAECanSwitchLayer` mode flag sent. This flag is exactly the opposite of the Apple event flag `kAECanSwitchLayer`. This is to provide a more convenient default, such as not supplying any mode (see `kOSANullMode` in the “Null Mode Flags” (page 106)) means to send events with `kAECanSwitchLayer`. Supplying the `kOSAModeCantSwitchLayer` mode flag will cause `AESend` to be called without `kAECanSwitchLayer`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDoRecord`

This mode flag may be passed to the functions [OSACompile](#) (page 30), [OSAExecute](#) (page 46), [OSALoadExecute](#) (page 63), [OSACompileExecute](#) (page 31), [OSADoScriptFile](#) (page 45), [OSAExecuteEvent](#) (page 47), and [OSADoEvent](#) (page 42) to indicate whether Apple events should be sent with the `kAEDontRecord` mode flag. This flag is exactly the opposite the Apple event flag `kAEDontRecord`. This is to provide a more convenient default, such as not supplying any mode (see `kOSANullMode` in the “Null Mode Flags” (page 106)) means to send events with `kAEDontRecord`. Supplying the `kOSAModeDoRecord` mode flag will cause `AESend` to be called without `kAEDontRecord`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeCompileIntoContext`

This is a mode flag for `OSACompile` (page 30) that indicates that a context should be created as the result of compilation. All handler definitions are inserted into the new context, and variables are initialized by evaluating their initial values in a null context (for example, they must be constant expressions).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeAugmentContext`

This is a mode flag for `OSACompile` (page 30) that indicates that the previous script ID (input to `OSACompile`) should be augmented with any new definitions in the `sourceData` parameter rather than replaced with a new script. This means that the previous script ID must designate a context. The presence of this flag causes the `kOSAModeCompileIntoContext` flag to be implicitly used, causing any new definitions to be initialized in a null context.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDisplayForHumans`

This mode flag may be passed to `OSADisplay` (page 41) or `OSADoScriptFile` (page 45) to indicate that output only need be human-readable, not re-compileable by `OSACompile` (page 30). If used, output may be arbitrarily "beautified", for example, quotes may be left off of string values, and long lists may have ellipses.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontStoreParent`

This mode flag may be passed to `OSAStore` (page 79) in the case where the `scriptID` parameter is a context. This causes the context to be saved, but not the context's parent context. When the stored context is loaded back in, the parent will be `kOSANullMode` (see the "Null Mode Flags" (page 106)).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDispatchToDirectObject`

This mode flag may be passed to `OSAExecuteEvent` (page 47) to cause the event to be dispatched to the direct object of the event. The direct object (or subject attribute if the direct object is a non-object specifier) will be resolved, and the resulting script object will be the recipient of the message. The context argument to `OSAExecuteEvent` will serve as the root of the lookup/resolution process.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeDontGetDataForArguments`

This mode flag may be passed to `OSAExecuteEvent` (page 47) to indicate that components do not have to get the data of object specifier arguments.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAModeFullyQualifyDescriptors`

This mode flag may be passed to [OSACoerceToDesc](#) (page 29) to indicate that the resulting descriptor should be fully qualified (i.e. should include the root application reference).

Available in Mac OS X v10.3 and later.

Declared in `OSA.h`.

Null Mode Flags

Indicate a function's default mode settings are to be used.

```
enum {
    kOSANullMode = 0,
    kOSAModeNull = 0
};
```

OSADebugStepKind

```
typedef UInt32 OSADebugStepKind;
enum {
    eStepOver = 0,
    eStepIn = 1,
    eStepOut = 2,
    eRun = 3
};
```

OSAProgramState

```
typedef UInt32 OSAProgramState;
enum {
    eNotStarted = 0,
    eRunnable = 1,
    eRunning = 2,
    eStopped = 3,
    eTerminated = 4
};
```

OSAScriptError Selectors

Define selectors used to retrieve information about script errors from the `OSAScriptError` function.

```

enum {
    kOSAErrorNumber = keyErrorNumber
};
enum {
    kOSAErrorMessage = keyErrorString
};
enum {
    kOSAErrorBriefMessage = 'errb'
};
enum {
    kOSAErrorApp = 'erap'
};
enum {
    kOSAErrorPartialResult = 'ptlr'
};
enum {
    kOSAErrorOffendingObject = 'erob'
};
enum {
    kOSAErrorExpectedType = 'errt'
};
enum {
    kOSAErrorRange = 'erng'
};

```

Constants

kOSAErrorNumber

This selector is used to determine the error number of a script error. These error numbers may be either system error numbers, or error numbers that are scripting component specific. The value of `desiredType` must be `typeShortInteger`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorMessage

This selector is used to determine the full error message associated with the error number. It should include the name of the application which caused the error, as well as the specific error that occurred. This selector is sufficient for simple error reporting (but see `kOSAErrorBriefMessage`). The value of `desiredType` must be `typeChar` or another text descriptor type.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorBriefMessage

This selector is used to determine a brief error message associated with the error number. This message should not mention the name of the application which caused the error, any partial results or offending object (see `kOSAErrorApp`, `kOSAErrorPartialResult`, and `kOSAErrorOffendingObject`). The value of `desiredType` must be `typeChar` or another text descriptor type.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorApp

This selector is used to determine which application actually got the error (if it was the result of an `AESend`). The value of `desiredType` must be `typeProcessSerialNumber` (for the PSN) or a text descriptor type such as `typeChar` (for the name).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorPartialResult

This selector is used to determine any partial result returned by an operation. If an `AESend` call failed, but a partial result was returned, then the partial result may be returned as an `AEDesc`. The value of `desiredType` must be `typeBest` (for the best type) or `typeWildcard` (for the default type).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorOffendingObject

This selector is used to determine any object which caused the error that may have been indicated by an application. The result is an `AEDesc`. The value of `desiredType` must be `typeObjectSpecifier`, `typeBest`, or `typeWildcard`. For some scripting components, including AppleScript, these three values are equivalent.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorRange

This selector is used to determine the source text range (start and end positions) of where the error occurred. The value of `desiredType` must be `typeOSAErrorRange`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

kOSAErrorExpectedType

This selector is used to determine the type expected by a coercion operation if a type error occurred.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Recording Constants

```
enum {
    kOSASelectStartRecording = 0x0501,
    kOSASelectStopRecording = 0x0502
};
```

Resume Dispatch Function Constants

Define constants used with the `OSASetResumeDispatchProc` function.

```
enum {
    kOSAUseStandardDispatch = kAEUseStandardDispatch
};
enum {
    kOSANoDispatch = kAENoDispatch
};
enum {
    kOSADontUsePhac = 0x0001
};
```

Constants

`kOSAUseStandardDispatch`

Used in the `resumeDispatchProc` parameter of [OSASetResumeDispatchProc](#) (page 74) and [OSAGetResumeDispatchProc](#) (page 56) to indicate that the event is dispatched using standard Apple event dispatching (the handler registered in the application with `AEInstallEventHandler` should be used).

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSANoDispatch`

Used in the `resumeDispatchProc` parameter of [OSASetResumeDispatchProc](#) (page 74) to tell the Apple Event Manager that the processing of the Apple event is complete and that no dispatching should occur.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSADontUsePhac`

Used in the `refCon` parameter of [OSASetResumeDispatchProc](#) (page 74) to dispatch the event using standard Apple event dispatching, except that the predispach handler should not be called. Used only in conjunction with `kOSAUseStandardDispatch`. This is useful when the predispach handler is used to lookup a context associated with an event's direct parameter and call [OSAExecuteEvent](#) (page 47) or [OSADoEvent](#) (page 42). Failure to bypass the predispach handler when resuming an event in this case would result in an infinite loop. (A predispach handler is called immediately before the Apple Event Manager dispatches an event.)

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

Script Document File Type

Defines the file type of script document files.

```
enum {
    kOSAFileType = 'osas'
};
```

Script Information Selectors

Specify which script information is set or returned.

```

enum {
    kOSAScriptIsModified = 'modi'
};
enum {
    kOSAScriptIsTypeCompiledScript = 'cscr'
};
enum {
    kOSAScriptIsTypeScriptValue = 'valu'
};
enum {
    kOSAScriptIsTypeScriptContext = 'cntx'
};
enum {
    kOSAScriptBestType = 'best'
};
enum {
    kOSACanGetSource = 'gsrc'
};
enum {
    kASHasOpenHandler = 'hsod'
};

```

Constants

`kOSAScriptIsModified`

This selector is used to determine whether there have been any changes since the script data was loaded or created. In Mac OS X, the AppleScript component returns a value of `false` if no changes have been made, and a value of `true` if changes may have been made. For more information, see the Version Notes section for the [OSAGetScriptInfo](#) (page 56) function.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeCompiledScript`

This selector is used to determine whether or not the script data is a compiled script. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeScriptValue`

This selector is used to determine whether or not the script data is a script value. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptIsTypeScriptContext`

This selector is used to determine whether or not the script data is a script context. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSAScriptBestType`

A descriptor type that you can pass to `OSACoerceToDesc`.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kOSACanGetSource`

This selector is used to determine whether a script has source associated with it that when given to `OSAGetSource`, the call will not fail. The selector returns a boolean.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

`kASHasOpenHandler`

This selector is used to query a context as to whether it contains a handler for the `kAEOpenDocuments` event. This allows "applets" to be distinguished from "droplets." [OSAGetScriptInfo \(page 56\)](#) returns false if there is no `kAEOpenDocuments` handler, and returns the error value `errOSAInvalidAccess` if the input is not a context.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

Source Constants

```
enum {
    kOSASelectGetSource = 0x0201
};
```

Source Style Constants

Identify script format styles used by the AppleScript component to display scripts.

```
enum {
    kASSourceStyleUncompiledText = 0,
    kASSourceStyleNormalText = 1,
    kASSourceStyleLanguageKeyword = 2,
    kASSourceStyleApplicationKeyword = 3,
    kASSourceStyleComment = 4,
    kASSourceStyleLiteral = 5,
    kASSourceStyleUserSymbol = 6,
    kASSourceStyleObjectSpecifier = 7,
    kASNumberOfSourceStyles = 8
};
```

Constants

`kASSourceStyleUncompiledText`
Script format style for uncompiled text.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleNormalText`
Script format style for normal text.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleLanguageKeyword`
Script format style for keywords of the AppleScript Language.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleApplicationKeyword`
Script format style for keywords of a scriptable application.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleComment`
Script format style for comment text.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleLiteral`
Script format style for literal text.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleUserSymbol`
A user-defined symbol, such as a variable or custom handler name.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASSourceStyleObjectSpecifier`
Deprecated.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

`kASNumberOfSourceStyles`
Deprecated. (The number of different format styles available.)
See the Discussion section for why you should not use this constant.
Available in Mac OS X v10.0 and later.
Declared in `AppleScript.h`.

Discussion

These constants are used to access specific styles in the style information used by the [ASCopySourceAttributes](#) (page 17), [ASSetSourceAttributes](#) (page 22), and [ASGetSourceStyleNames](#) (page 19) functions (and the deprecated functions [ASGetSourceStyles](#) (page 119) and [ASSetSourceStyles](#) (page 119)).

The order of the style information corresponds to the order of the constants. For example, the first dictionary in the array returned by [ASCopySourceAttributes](#) (page 17) (at position `kASSourceStyleUncompiledText`) describes the style for uncompiled text. However, you should not rely on there being any specific number of dictionaries (such as `kASNumberOfSourceStyles`) in the returned array—instead, count the number of items in the array before accessing any of them.

Declared In

`AppleScript.h`

typeAppleScript

Define descriptor types for the AppleScript instance of the Open Scripting Architecture type.


```
enum {
    typeAppleScript = 'ascr',
    kAppleScriptSubtype = typeAppleScript,
    typeASStorage = typeAppleScript
};
```

Constants

`kAppleScriptSubtype`

Defines the Component Manager subtype for the AppleScript component.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

`typeASStorage`

Defines the AppleScript constant for storage descriptor records.

Available in Mac OS X v10.0 and later.

Declared in `AppleScript.h`.

typeOSAErrorRange

Defines the descriptor type for an error range.

```
enum {
    typeOSAErrorRange = 'erng'
};
```

typeOSAGenericStorage

Defines the descriptor type for generic storage descriptor records.

```
enum {
    typeOSAGenericStorage = kOSAScriptResourceType
};
```

Constants

`typeOSAGenericStorage`

Default type given to [OSAStore](#) (page 79), which creates "generic" loadable script data descriptors.

Available in Mac OS X v10.0 and later.

Declared in `OSA.h`.

Declared In

`OSA.h`

typeStatementRange

```
enum {
    typeStatementRange = 'srng'
};
```

Weekdays

```
enum {
    cWeekday = pASWeekday,
    cSunday = 'sun ',
    cMonday = 'mon ',
    cTuesday = 'tue ',
    cWednesday = 'wed ',
    cThursday = 'thu ',
    cFriday = 'fri ',
    cSaturday = 'sat ',
    pASQuote = 'quot',
    pASSeconds = 'secs',
    pASMinutes = 'min ',
    pASHours = 'hour',
    pASDays = 'days',
    pASWeeks = 'week',
    cWritingCodeInfo = 'citl',
    pScriptCode = 'pscd',
    pLangCode = 'plcd',
    kASMagicTellEvent = 'tell',
    kASMagicEndTellEvent = 'tend'
};
```

Result Codes

The most common result codes returned by Open Scripting Architecture are listed in Table 1-1. Open Scripting Architecture may also return the result codes `noErr (0)`, and `badComponentInstance (-2147450879)`.

Result Code	Value	Description
<code>errOSACantCoerce</code>	-1700	A value can't be coerced to the desired type. Available in Mac OS X v10.0 and later.
<code>OSAMissingParameter</code>	-1701	A parameter is missing for a function invocation. Available in Mac OS X v10.0 and later.
<code>errOSACorruptData</code>	-1702	Some data could not be read. Available in Mac OS X v10.0 and later.
<code>errOSATypeError</code>	-1703	Same as <code>errAEWrongDataType</code> ; wrong descriptor type. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
OSAMessageNotUnderstood	-1708	A message was sent to an object that didn't handle it. Available in Mac OS X v10.0 and later.
OSAUndefinedHandler	-1717	A function to be returned doesn't exist. Available in Mac OS X v10.0 and later.
OSAIllegalIndex	-1719	An index was out of range. Specialization of <code>errOSACantAccess</code> . Available in Mac OS X v10.0 and later.
OSAIllegalRange	-1720	The specified range is illegal. Specialization of <code>errOSACantAccess</code> . Available in Mac OS X v10.0 and later.
OSAParameterMismatch	-1721	The wrong number of parameters were passed to the function, or a parameter pattern cannot be matched. Available in Mac OS X v10.0 and later.
OSAIllegalAccess	-1723	A container can not have the requested object. Available in Mac OS X v10.0 and later.
errOSACantAccess	-1728	An object is not found in a container. Available in Mac OS X v10.0 and later.
errOSARecordingIsAlreadyOn	-1732	Recording is already on. Available in Mac OS X v10.0 and later.
errOSASystemError	-1750	Scripting component error. Available in Mac OS X v10.0 and later.
errOSAInvalidID	-1751	Invalid script id. Available in Mac OS X v10.0 and later.
errOSABadStorageType	-1752	Script doesn't seem to belong to AppleScript. Available in Mac OS X v10.0 and later.
errOSAScriptError	-1753	Script error. Available in Mac OS X v10.0 and later.
errOSABadSelector	-1754	Invalid selector given. Available in Mac OS X v10.0 and later.
errOSASourceNotAvailable	-1756	Invalid access. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errOSANoSuchDialect	-1757	Source not available. Available in Mac OS X v10.0 and later.
errOSADataFormatObsolete	-1758	No such dialect. Available in Mac OS X v10.0 and later.
errOSADataFormatTooNew	-1759	Data couldn't be read because its format is obsolete. Available in Mac OS X v10.0 and later.
errOSAComponentMismatch	-1761	Parameters are from two different components. Available in Mac OS X v10.0 and later.
errOSACantOpenComponent	-1762	Can't connect to system with that ID. Available in Mac OS X v10.0 and later.
errOSAGeneralError	-2700	No actual error code is to be returned. Available in Mac OS X v10.0 and later.
errOSADivideByZero	-2701	An attempt to divide by zero was made. Available in Mac OS X v10.0 and later.
errOSANumericOverflow	-2702	An integer or real value is too large to be represented. Available in Mac OS X v10.0 and later.
errOSACantLaunch	-2703	An application can't be launched, or when it is, remote and program linking is not enabled. Available in Mac OS X v10.0 and later.
errOSAAppNotHighLevelEventAware	-2704	An application can't respond to Apple events. Available in Mac OS X v10.0 and later.
errOSACorruptTerminology	-2705	An application's terminology resource is not readable. Available in Mac OS X v10.0 and later.
errOSAStackOverflow	-2706	The runtime stack overflowed. Available in Mac OS X v10.0 and later.
errOSAInternalTableOverflow	-2707	A runtime internal data structure overflowed. Available in Mac OS X v10.0 and later.
errOSADataBlockTooLarge	-2708	An intrinsic limitation is exceeded for the size of a value or data structure. Available in Mac OS X v10.0 and later.
errOSACantGetTerminology	-2709	Can't get the event dictionary. Available in Mac OS X v10.0 and later.

Result Code	Value	Description
errOSACantCreate	-2710	Can't make class <class identifier>. Available in Mac OS X v10.0 and later.
OSASyntaxError	-2740	A syntax error occurred. Available in Mac OS X v10.0 and later.
OSASyntaxTypeError	-2741	Another form of syntax was expected. Available in Mac OS X v10.0 and later.
OSATokenTooLong	-2742	A name or number is too long to be parsed. Available in Mac OS X v10.0 and later.
OSADuplicateParameter	-2750	A formal parameter, local variable, or instance variable is specified more than once. Available in Mac OS X v10.0 and later.
OSADuplicateProperty	-2751	A formal parameter, local variable, or instance variable is specified more than once. Available in Mac OS X v10.0 and later.
OSADuplicateHandler	-2752	More than one handler is defined with the same name in a scope where the language doesn't allow it. Available in Mac OS X v10.0 and later.
OSAUndefinedVariable	-2753	A variable is accessed that has no value. Available in Mac OS X v10.0 and later.
OSAINconsistentDeclarations	-2754	A variable is declared inconsistently in the same scope, such as both local and global. Available in Mac OS X v10.0 and later.
OSAControlFlowError	-2755	An illegal control flow occurs in an application. For example, there is no catcher for the throw, or there was a non-lexical loop exit. Available in Mac OS X v10.0 and later.
OSAIlllegalAssign	-10003	An object can never be set in a container Available in Mac OS X v10.0 and later.
errOSACantAssign	-10006	An object cannot be set in a container. Available in Mac OS X v10.0 and later.

Deprecated Open Scripting Architecture Functions

A function identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in Mac OS X v10.5

ASGetSourceStyles

Gets the script format styles currently used by the AppleScript component to display scripts. (Deprecated in Mac OS X v10.5. Use [ASGetSourceStyleNames](#) (page 19) instead.)

```
OSAEError ASGetSourceStyles (
    ComponentInstance scriptingComponent,
    STHandle *resultingSourceStyles
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

resultingSourceStyles

A pointer to a handle to a style element array defined by the TextEdit data type `TEStyleTable` that defines the styles used for different kinds of AppleScript terms.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `ASGetSourceStyles` function returns a style element array that defines the styles used for AppleScript terms. You can use the index constants described in [“Source Style Constants”](#) (page 111) to identify individual styles returned in the *resultingSourceStyles* parameter. Other AppleScript dialects may define additional styles. When you have finished using the style element array, you must dispose of it.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

`AppleScript.h`

ASSetSourceStyles

Sets the script format styles used by the AppleScript component to display scripts. (Deprecated in Mac OS X v10.5. Use [ASSetSourceAttributes](#) (page 22) instead.)

Deprecated Open Scripting Architecture Functions

```
OSAEError ASSetSourceStyles (
    ComponentInstance scriptingComponent,
    SHandle sourceStyles
);
```

Parameters

scriptingComponent

A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent`.

sourceStyles

A handle to a style element array defined by the TextEdit data type `TStyleTable` that defines the styles used for different kinds of AppleScript terms. The style for each kind of term should be identified according to the index constants listed in [“Source Style Constants”](#) (page 111).

Return Value

A result code. See [“Result Codes”](#) (page 114).

Discussion

The `ASSetSourceStyles` function sets the script format styles used to display scripts. If you pass a `NULL` handle in the *sourceStyles* parameter, the AppleScript component uses its default styles.

After you have set the script format styles, you must dispose of the style element array you used to specify them.

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Declared In

AppleScript.h

OSAGetAppTerminology

Gets one or more scripting terminology resources from the specified file. (Deprecated in Mac OS X v10.5. Use [OSACopyScriptingDefinition](#) (page 33) instead.)

```
OSAEError OSAGetAppTerminology (
    ComponentInstance scriptingComponent,
    SInt32 modeFlags,
    FSSpec *fileSpec,
    short terminologyID,
    Boolean *didLaunch,
    AEDesc *terminologyList
);
```

Parameters

scriptingComponent

Identifies the current scripting component. See the Component Manager documentation for a description of the `ComponentInstance` data type.

modeFlags

Information for use by the scripting component. No mode flags are applicable for this function, so pass the value `kOSAModeNull`.

Deprecated Open Scripting Architecture Functions

fileSpec

Specifies the file to search. See the File Manager documentation for a description of the `FSSpec` data type.

terminologyID

A dialect code obtained from a previous call to the `OSAGetDialectInfo` function or the `OSAGetCurrentDialect` function.

didLaunch

On return, has the value `true` if the application's scripting size resource or plist flags indicate that it has a dynamic terminology (in which case, the application will have been launched).

terminologyList

On return, a descriptor list containing zero or more terminology resources. See Apple Event Manager Reference for a description of the `AEDesc` data type.

Return Value

A result code. See [“Result Codes”](#) (page 114).

Availability

Available in Mac OS X v10.0 and later.

Deprecated in Mac OS X v10.5.

Not available to 64-bit applications.

Declared In

`ASDebugging.h`

Document Revision History

This table describes the changes to *Open Scripting Architecture Reference*.

Date	Notes
2007-05-07	Added documentation for new functions and other changes in Mac OS X version 10.5.
	The new functions are ASCopySourceAttributes (page 17), ASSetSourceAttributes (page 22), OSACopyDisplayString (page 32), and OSACopySourceString (page 34).
	The functions ASGetSourceStyles (page 119) and ASSetSourceStyles (page 119) are deprecated in Mac OS X version 10.5; use ASCopySourceAttributes (page 17) and ASSetSourceAttributes (page 22) instead.
	Removed undocumented constants that can be used with <code>CallComponentFunction</code> , such as <code>kOSASelectLoad</code> and <code>kASSelectSetSourceStyles</code> , because they have easier-to-use function equivalents, such as OSALoad (page 62) and ASSetSourceStyles (page 119) (though the latter is deprecated in Mac OS X version 10.5, in favor of ASSetSourceAttributes (page 22)).
	For the function ASInit (page 20) and the constants in “ Default Initialization Values ” (page 92) that you use with ASInit , noted that starting in Mac OS X version 10.5, heap size parameter values are ignored—AppleScript’s heap will grow as large as needed.
	Removed the description for the <code>OSACopyScript</code> function because it has never been defined in a public header.
	Made minor changes to the Discussion sections for the functions OSADoScript (page 44) and OSADoScriptFile (page 45), including that for OSADoScriptFile , the Discussion now correctly refers to OSAExecute (page 46), rather than OSAExecuteEvent (page 47).
	In “ Source Style Constants ” (page 111), noted that you should not use the constant <code>kASNumberOfSourceStyles</code> to determine the number of style items used by the ASCopySourceAttributes (page 17), ASSetSourceAttributes (page 22), and ASGetSourceStyleNames (page 19) functions (and the deprecated functions ASGetSourceStyles (page 119) and ASSetSourceStyles (page 119)).
2005-07-07	Moved some functions to more appropriate groups to make them easier to find.

REVISION HISTORY

Document Revision History

Date	Notes
	Provided the correct descriptions for <code>kOSANoDispatch</code> and <code>kOSADontUsePhac</code> in “Resume Dispatch Function Constants” (page 108).
2005-04-29	Updated for Mac OS X v10.4. Filled in missing error code descriptions and made minor text corrections.
	Added description for function <code>OSACopyScriptingDefinition</code> (page 33), new in Mac OS X version 10.4 (v10.4).
	Added constant <code>kASNumericStrings</code> to “Considerations Flags” (page 90) and constants <code>kASNumericStringsConsiderMask</code> and <code>kASNumericStringsIgnoreMask</code> to “Considerations Bit Masks” (page 91); constants are new in Mac OS X v10.4.
	Added more stringent warning not to use the OSA debugging functions listed in “Deprecated Functions” (page 16).
	Added note to “Current Dialect Constants” (page 92) that the constants are not particularly useful because no currently available OSA languages support them.
	Made minor revision to Introduction.
	Deleted a duplicate entry for the error code constant <code>errOSARecordingIsAlreadyOn</code> . (The entry with the error number -1760 was incorrect.)
	Added Version Notes sections to <code>OSASetScriptInfo</code> (page 75) and <code>OSAGetScriptInfo</code> (page 56) to clarify use of the <code>selector</code> parameter in Mac OS X.
	Noted that the functions <code>AS SetProperty</code> (page 22) and <code>ASGetAppTerminology</code> (page 18) are obsolete and only available for backward compatibility, and that you should use <code>OSASetProperty</code> (page 73) and <code>OSAGetAppTerminology</code> (page 120) instead.
2003-08-21	Incorporated existing OSA reference documentation.
2003-07-31	Added descriptions for the following functions: <code>OSADoScriptFile</code> (page 45), <code>OSALoadExecuteFile</code> (page 64), <code>OSALoadFile</code> (page 65), <code>OSAStoreFile</code> (page 80)
	Moved <code>OSADebugger</code> functions to “Deprecated Functions” (page 16) and marked them as unsupported.
2003-01-01	Added comments available in header file.
	Updated Result Code section.
2001-07-01	Last version of this document.

Index

A

ASCopySourceAttributes [function 17](#)
ASGetAppTerminology [function 18](#)
ASGetHandler [function 18](#)
ASGetProperty [function 19](#)
ASGetSourceStyleNames [function 19](#)
ASGetSourceStyles [function \(Deprecated in Mac OS X v10.5\) 119](#)
ASInit [function 20](#)
ASSetHandler [function 21](#)
ASSetProperty [function 22](#)
ASSetSourceAttributes [function 22](#)
ASSetSourceStyles [function \(Deprecated in Mac OS X v10.5\) 119](#)

C

cClosure [88](#)
cCoercion [88](#)
cHandleBreakpoint [88](#)
Component Flags [89](#)
Considerations Bit Masks [91](#)
Considerations Flags [90](#)
cString [92](#)
Current Dialect Constants [92](#)

D

Date and Time Constants [92](#)
Default Initialization Values [92](#)
Dialect Descriptor Constants [94](#)
DisposeOSAActiveUPP [function 23](#)
DisposeOSACreateAppleEventUPP [function 23](#)
DisposeOSASendUPP [function 23](#)

E

errOSAAppNotHighLevelEventAware [constant 116](#)
errOSABadSelector [constant 115](#)
errOSABadStorageType [constant 115](#)
errOSACantAccess [constant 115](#)
errOSACantAssign [constant 117](#)
errOSACantCoerce [constant 114](#)
errOSACantCreate [constant 117](#)
errOSACantGetTerminology [constant 116](#)
errOSACantLaunch [constant 116](#)
errOSACantOpenComponent [constant 116](#)
errOSACComponentMismatch [constant 116](#)
errOSACorruptData [constant 114](#)
errOSACorruptTerminology [constant 116](#)
errOSADataBlockTooLarge [constant 116](#)
errOSADataFormatObsolete [constant 116](#)
errOSADataFormatTooNew [constant 116](#)
errOSADivideByZero [constant 116](#)
errOSAGeneralError [constant 116](#)
errOSAInternalTableOverflow [constant 116](#)
errOSAInvalidID [constant 115](#)
errOSANoSuchDialect [constant 116](#)
errOSANumericOverflow [constant 116](#)
errOSARecordingIsAlreadyOn [constant 115](#)
errOSAScriptError [constant 115](#)
errOSASourceNotAvailable [constant 115](#)
errOSAStackOverflow [constant 116](#)
errOSASystemError [constant 115](#)
errOSATypeError [constant 114](#)

G

Generic Scripting Component Selectors [95](#)
GenericID [data type 85](#)
Global Properties [95](#)

I

InvokeOSAActiveUPP function 24
 InvokeOSACreateAppleEventUPP function 24
 InvokeOSASendUPP function 25

K

kAppleScriptSubtype constant 113
 kASAdd 95
 kASAnd 95
 kASDefaultMaxHeapSize constant 93
 kASDefaultMaxStackSize constant 93
 kASDefaultMinHeapSize constant 93
 kASDefaultMinStackSize constant 93
 kASDefaultPreferredHeapSize constant 93
 kASDefaultPreferredStackSize constant 93
 kASErrorEventCode 96
 kASHasOpenHandler constant 111
 kASNumberOfSourceStyles constant 112
 kASNumericStrings constant 90
 kASNumericStringsConsiderMask constant 91
 kASNumericStringsIgnoreMask constant 91
 kASSourceStyleApplicationKeyword constant 112
 kASSourceStyleComment constant 112
 kASSourceStyleLanguageKeyword constant 111
 kASSourceStyleLiteral constant 112
 kASSourceStyleNormalText constant 111
 kASSourceStyleObjectSpecifier constant 112
 kASSourceStyleUncompiledText constant 111
 kASSourceStyleUserSymbol constant 112
 kASStartLogEvent 96
 kDialectBundleResType 96
 keyAETarget 97
 keyAppHandledCoercion 97
 keyASPrepositionAt 97
 keyASPrepositionOver 98
 keyOSADialectCode constant 94
 keyOSADialectLangCode constant 94
 keyOSADialectName constant 94
 keyOSADialectScriptCode constant 94
 keyOSASourceEnd 98
 keyOSASourceEnd constant 98
 keyOSASourceStart 98
 keyOSASourceStart constant 99
 keyProcedureName 99
 keyProgramState 99
 kGenericComponentVersion 99
 kGenericComponentVersion constant 99
 kOSACanGetSource constant 111
 kOSAComponentType 100
 kOSAComponentType constant 100
 kOSADontUsePhac constant 109
 kOSAErrorApp constant 107
 kOSAErrorBriefMessage constant 107
 kOSAErrorExpectedType constant 108
 kOSAErrorMessage constant 107
 kOSAErrorNumber constant 107
 kOSAErrorOffendingObject constant 108
 kOSAErrorPartialResult constant 108
 kOSAErrorRange constant 108
 kOSAGenericScriptingComponentSubtype 100
 kOSAModeAlwaysInteract constant 104
 kOSAModeAugmentContext constant 105
 kOSAModeCanInteract constant 104
 kOSAModeCantSwitchLayer constant 104
 kOSAModeCompileIntoContext constant 105
 kOSAModeDispatchToDirectObject constant 105
 kOSAModeDisplayForHumans constant 105
 kOSAModeDontDefine 100
 kOSAModeDontDefine constant 100
 kOSAModeDontGetDataForArguments constant 105
 kOSAModeDontReconnect constant 104
 kOSAModeDontStoreParent constant 105
 kOSAModeDoRecord constant 104
 kOSAModeFullyQualifyDescriptors constant 106
 kOSAModeNeverInteract constant 103
 kOSAModePreventGetSource constant 103
 kOSANoDispatch constant 109
 kOSANullScript 100
 kOSAScriptRecordedText 101
 kOSAScriptBestType constant 110
 kOSAScriptIsModified constant 110
 kOSAScriptIsTypeCompiledScript constant 110
 kOSAScriptIsTypeScriptContext constant 110
 kOSAScriptIsTypeScriptValue constant 110
 kOSAScriptResourceType 101
 kOSAScriptResourceType constant 101
 kOSASelectComponentSpecificStart 101
 kOSASelectComponentSpecificStart constant 101
 kOSASelectCopyScript 102
 kOSASuite 102
 kOSASupportsAECOercion constant 89
 kOSASupportsAESending constant 89
 kOSASupportsCompiling constant 89
 kOSASupportsConvenience constant 89
 kOSASupportsDialects constant 90
 kOSASupportsEventHandling constant 90
 kOSASupportsGetSource constant 89
 kOSASupportsRecording constant 89
 kOSAUseStandardDispatch constant 109

M

Mode Flags 102

N

NewOSAActiveUPP function 25
 NewOSACreateAppleEventUPP function 25
 NewOSASendUPP function 26
 Null Mode Flags 106

O

OSAActiveProcPtr callback 81
 OSAActiveUPP data type 86
 OSAAddStorageType function 26
 OSAAvailableDialectCodeList function 27
 OSAAvailableDialects function 27
 OSACoerceFromDesc function 28
 OSACoerceToDesc function 29
 OSACompile function 30
 OSACompileExecute function 31
 OSAControlFlowError constant 117
 OSACopyDisplayString function 32
 OSACopyID function 33
 OSACopyScriptingDefinition function 33
 OSACopySourceString function 34
 OSACreateAppleEventProcPtr callback 81
 OSACreateAppleEventUPP data type 86
 OSADebugCallFrameRef data type 86
 OSADebuggerCreateSession function 35
 OSADebuggerDisposeCallFrame function 35
 OSADebuggerDisposeSession function 36
 OSADebuggerGetBreakpoint function 36
 OSADebuggerGetCallFrameState function 37
 OSADebuggerGetCurrentCallFrame function 37
 OSADebuggerGetDefaultBreakpoint function 37
 OSADebuggerGetPreviousCallFrame function 38
 OSADebuggerGetSessionState function 38
 OSADebuggerGetStatementRanges function 39
 OSADebuggerGetVariable function 39
 OSADebuggerSessionStep function 39
 OSADebuggerSetBreakpoint function 40
 OSADebuggerSetVariable function 40
 OSADebugSessionRef data type 87
 OSADebugStepKind 106
 OSADisplay function 41
 OSADispose function 42
 OSADoEvent function 42
 OSADoScript function 44

OSADoScriptFile function 45
 OSADuplicateHandler constant 117
 OSADuplicateParameter constant 117
 OSADuplicateProperty constant 117
 OSAError data type 85
 OSAExecute function 46
 OSAExecuteEvent function 47
 OSAGenericToRealID function 48
 OSAGetActiveProc function 49
 OSAGetAppTerminology function (Deprecated in Mac OS X v10.5) 120
 OSAGetCreateProc function 50
 OSAGetCurrentDialect function 50
 OSAGetDefaultScriptingComponent function 51
 OSAGetDialectInfo function 51
 OSAGetHandler function 52
 OSAGetHandlerNames function 53
 OSAGetProperty function 54
 OSAGetPropertyNames function 55
 OSAGetResumeDispatchProc function 56
 OSAGetScriptInfo function 56
 OSAGetScriptingComponent function 57
 OSAGetScriptingComponentFromStored function 58
 OSAGetSendProc function 59
 OSAGetSource function 59
 OSAGetStorageType function 60
 OSAGetSysTerminology function 61
 OSAID data type 84
 OSAIllegalAccess constant 115
 OSAIllegalAssign constant 117
 OSAIllegalIndex constant 115
 OSAIllegalRange constant 115
 OSAInconsistentDeclarations constant 117
 OSALoad function 62
 OSALoadExecute function 63
 OSALoadExecuteFile function 64
 OSALoadFile function 65
 OSAMakeContext function 66
 OSAMessageNotUnderstood constant 115
 OSAMissingParameter constant 114
 OSAParameterMismatch constant 115
 OSAProgramState 106
 OSARealToGenericID function 66
 OSARemoveStorageType function 67
 OAScriptError function 68
 OAScriptError Selectors 106
 OAScriptingComponentName function 69
 OSASendProcPtr callback 83
 OSASendUPP data type 86
 OSASetActiveProc function 69
 OSASetCreateProc function 70
 OSASetCurrentDialect function 71
 OSASetDefaultScriptingComponent function 71

OSASetDefaultTarget **function** [72](#)
OSASetHandler **function** [73](#)
OSASetProperty **function** [73](#)
OSASetResumeDispatchProc **function** [74](#)
OSASetScriptInfo **function** [75](#)
OSASetSendProc **function** [76](#)
OSAStartRecording **function** [77](#)
OSAStopRecording **function** [78](#)
OSAStore **function** [79](#)
OSAStoreFile **function** [80](#)
OSASyntaxError **constant** [117](#)
OSASyntaxTypeError **constant** [117](#)
OSATokenTooLong **constant** [117](#)
OSAUndefinedHandler **constant** [115](#)
OSAUndefinedVariable **constant** [117](#)

R

Recording Constants [108](#)
Resume Dispatch Function Constants [108](#)

S

Script Document File Type [109](#)
Script Information Selectors [109](#)
ScriptingComponentSelector **data type** [85](#)
Source Constants [111](#)
Source Style Constants [111](#)
StatementRange **structure** [85](#)

T

typeAppleScript [112](#)
typeASStorage **constant** [113](#)
typeOSADialectInfo **constant** [94](#)
typeOSAErrorRange [113](#)
typeOSAGenericStorage [113](#)
typeOSAGenericStorage **constant** [113](#)
typeStatementRange [114](#)

W

Weekdays [114](#)