CHAPTER 10

Scripting Components

This chapter describes how your application can use the Component Manager and scripting components to manipulate and execute scripts.

Before you read this chapter, you should read the chapter “Introduction to Scripting” in this book and the chapters about the Apple Event Manager that are relevant to your application.

Your application can use the standard scripting component data structures and routines described in this chapter to manipulate scripts written in any scripting language based on the Open Scripting Architecture (OSA). Your application need not be scriptable or recordable to use these routines. However, if your application is scriptable, you can easily make it capable of manipulating and executing scripts that control its own behavior.

The first section in this chapter describes how to establish a connection with a scripting component. The next two sections provide

■ examples of how to use the standard scripting component routines
■ information for developers of scripting components

The section “Scripting Components Reference” describes, in addition to the standard scripting component routines, routines provided by the AppleScript component, routines provided by the generic scripting component, and routines called by scripting components.

If you are developing a scripting component, you should also read the instructions for creating components in the chapter “Component Manager” in *Inside Macintosh: More Macintosh Toolbox.*

Connecting to a Scripting Component

To manipulate and execute scripts written in different scripting languages, your application can use Component Manager routines either to open a connection with each corresponding scripting component individually or to open a single connection with the generic scripting component. The generic scripting component, in turn, attempts to open connections dynamically with the appropriate scripting component for each script. By opening a connection with the generic scripting component, your application can load and execute scripts created by any scripting component that is registered with the Component Manager on the current computer.

In general, you should use the generic scripting component to execute and manipulate existing scripts and a specific scripting component when you create new scripts. When you call *OSACompile* or *OSAStartRecording*, the generic scripting component examines the script ID to determine which scripting component to use. If instead of a script ID you pass the constant *kOSANullScript* to these routines, the generic scripting component uses its current default scripting component. Each instance of the generic scripting component has its own default scripting component. 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.
CHAPTER 10

Scripting Components

The generic scripting component provides routines you can use to get and set the default scripting component, determine which scripting component created a particular script, and perform other useful tasks when you are using multiple scripting components. See the section “Generic Scripting Component Routines,” which begins on page 10-84, for descriptions of these routines.

You can use the Component Manager function OpenComponent to open a connection to a scripting component you specify with the component identifier returned by the FindNextComponent function. You can also use the OpenDefaultComponent function to open a scripting component without calling the FindNextComponent function.

The OpenComponent and OpenDefaultComponent functions return a component instance. This value identifies your application’s connection to a component. You must supply this value whenever you call a standard scripting component routine.

Note
Your application may maintain several connections to a single component, or it may have connections to several components at the same time. Because some scripting components (including the current version of AppleScript) can execute only one script at a time per component instance, a multithreaded application must provide a separate component instance for each script that it compiles or executes while it is simultaneously executing other scripts.

The Component Manager type code for scripting components that support the routines described in this chapter is 'osa ', and the subtype code for the generic scripting component is 'scpt'.

CONST
    kOSAComponentType = 'osa ';
    kOSAGenericscriptingComponentSubtype = 'scpt';

You can open a connection to a scripting component by calling the OpenDefaultComponent function, which returns a component instance. For example, this code opens a connection with the generic scripting component and stores the returned value in an application-defined variable:

VAR
    gScriptingComponent: ComponentInstance;

    {open connection to generic scripting component}
    gScriptingComponent := OpenDefaultComponent(kOSAComponentType,
                                          kOSAGenericscriptingComponentSubtype);

The generic scripting component in turn opens connections with other scripting components as necessary. The generic scripting component provides routines you can use to get instances of other scripting components when you want to use component-specific routines.
Chapter 10

Scripting Components

It is also possible to open an explicit connection directly with a specific scripting component such as AppleScript:

VAR gScriptingComponent: ComponentInstance;
{open connection to AppleScript component}
gScriptingComponent := OpenDefaultComponent(kOSAComponentType,
  kAppleScriptSubtype);

The scripting component routines described in this chapter include eight groups of optional routines that scripting components can support. If necessary, you can use the FindNextComponent function and other Component Manager routines to find a scripting component that supports a specific group of routines or to determine whether a particular scripting component supports a specific group of routines.

When you call FindNextComponent, you can provide, in a component description record (a data structure of type ComponentDescription), information about the scripting component you wish to find. The flag bits in the componentFlags field of a component description record provide this information. To find a scripting component that supports a specific group of optional routines, you can specify one or more of these constants in the componentFlags field:

CONST
  kOSASupportsCompiling = $0002;
  kOSASupportsGetSource = $0004;
  kOSASupportsAECoercion = $0008;
  kOSASupportsAESending = $0010;
  kOSASupportsRecording = $0020;
  kOSASupportsConvenience = $0040;
  kOSASupportsDialects = $0080;
  kOSASupportsEventHandling = $0100;

The routines that correspond to these constants are described in “Optional Scripting Component Routines,” which begins on page 10-46.

Note

Although the generic scripting component supports all the scripting component routines represented by these flags, the support it can actually provide depends on the individual components with which it opens connections.

Listing 10-1 shows how you can use these flags and the FindNextComponent function to locate a scripting component with specific characteristics. The componentFlags field of the component description record passed to FindNextComponent specifies the flags kOSASupportsCompiling and kOSASupportsGetSource. Because the componentFlagsMask field also specifies these flags, the FindNextComponent function locates a scripting component that supports these routines, regardless of whether or not it supports any others. The FindNextComponent function returns a
CHAPTER 10

Scripting Components

component identifier that you can then use to get more information about
the component or to open it.

Listing 10-1  Locating a scripting component that supports specific optional routines

FUNCTION MyConnectToScripting (VAR scriptingComponent: ComponentInstance)
  : OSAError;
VAR
descr, descr2: componentDescription;
comp: component;
myErr: OSErr;
BEGIN
  {fill in the fields of the component description record}
  {first specify component type, subtype, and manufacturer}
descr.componentType := kOSAComponentType; {must be scripting component}
descr.componentSubType := OSType(0); {any OSA component matching spec}
descr.componentManufacturer := OSType(0); {don't care about manufacturer}

  {specify component flags and flags mask}
descr.componentFlags := kOSASupportsCompiling + kOSASupportsGetSource;
descr.componentFlagsMask :=
  kOSASupportsCompiling + kOSASupportsGetSource;

  {locate and open the specified component}
  comp := FindNextComponent(Component(0), descr); {0 indicates all }  
    { registered components }  
    { will be searched}
  
  {check whether the found component is the generic scripting component; }  
  { if so, skip it and find the next matching component}
  myErr := GetComponentInfo(comp, descr2, NIL, NIL, NIL);
  IF descr2.componentSubType = kOSAGenericScriptingComponentSubtype THEN
    comp := FindNextComponent(comp, descr);
  IF comp = 0 THEN
    MyConnectToScripting := kComponentNotFound
  ELSE
    scriptingComponent := OpenComponent(comp);
    IF scriptingComponent = 0 THEN
      MyConnectToScripting := kComponentNotFound
    ELSE
      MyConnectToScripting := noErr;
    END;
  END;
END;
CHAPTER 10

Scripting Components

Because the generic scripting component supports all the standard scripting component routines, the MyConnectToScripting function in Listing 10-1 checks whether the found component is the generic scripting component and, if so, skips it. If for any reason FindNextComponent can’t locate and open a scripting component that supports the specified routines, MyConnectToScripting returns the application-defined constant kComponentNotFound.

For more information about locating and opening components with specific characteristics, see the chapter “Component Manager” in Inside Macintosh: More Macintosh Toolbox.

Using Scripting Component Routines

The following sections describe how to use some of the standard scripting component routines to manipulate and execute scripts from within your application. For an overview of these routines, see “Manipulating and Executing Scripts,” which begins on page 7-22.

The first section describes how to compile and execute source data for a script. The remaining sections describe how you can use scripting component routines to

- get a handle to a compiled script and save the data as a resource
- load and execute a previously saved and compiled script
- load, modify, recompile, and save a compiled script
- redirect Apple events to handlers in script contexts
- supply a resume dispatch function
- supply an alternative active function
- supply alternative send and create functions
- record Apple events in compiled scripts and display equivalent source data to the user

Compiling and Executing Source Data

This section describes how you can use scripting component routines to obtain source data from users, compile the source data, and execute the compiled script. To create and execute a script using the Script Editor application, a user can type the script, then click the Run button to execute it. Your application can provide similar capabilities.

To allow users to write a new script and then execute it, your application must use scripting component routines to compile and execute the source data. To compile source data in a new script with a new script ID, pass the constant kOSANullScript (rather than an existing script ID) in the last parameter of the OSACompile function. This causes OSACompile to return a new script ID in the same parameter.

To execute a compiled script, your application must specify, in addition to the script ID for the compiled script, a script context: either the corresponding scripting component’s
default context or a script ID for the global context created by that scripting component. Script contexts maintain state information for the execution of scripts. Your application can use script contexts to control the binding of variables used in scripts that it executes. For example, if your application saves its own global context and reuses it every time a script is executed, the binding of variables used in the script is maintained after the user restarts the computer. If your application does not specify a script context, the AppleScript component uses a single default context whenever it executes the script. A scripting component’s default context binds the variables used in the script only until the user quits the application.

To specify a scripting component’s default context, pass the constant kOSANullScript in the third parameter of the OSAExecute function; to specify some other global context, pass its script ID in the third parameter.

The MyDoNewScript procedure in Listing 10-2 allows a user to type a script in the appropriate scripting language, then compiles the script, executes the compiled script using a global context provided by the application, and displays the result to the user.

The MyDoNewScript procedure begins by calling the OAScriptingComponentName function to obtain the name of the scripting component specified by gScriptingComponent. This name is passed to the application-defined function MyGetUserScriptText.

Note

If you are using the generic scripting component, you can use the OSAGetDefaultComponent function 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 OAScriptingComponentName to obtain the default scripting component’s name. For more information about the default scripting component and routines you can use with the generic scripting component, see “Generic Scripting Component Routines,” which begins on page 10-84.

The MyGetUserScriptText function displays the name of the scripting language to use in a script-editing window or message box that allows the user to type and execute a new script. After it obtains the source data for the new script, the MyDoNewScript procedure sets the scriptID variable to kOSANullScript. The procedure then passes the source data and scriptID to the OSACompile function. When the script ID passed to OSACompile is kOSANullScript, OSACompile returns, in the same parameter, a new script ID for the resulting compiled script. The MyDoNewScript procedure then passes the new script ID to the OSAExecute function.

In addition to a component instance and the script ID for the compiled script to be executed, OSAExecute takes a script ID for a script context and a parameter that contains the mode flags, if any, for script execution. In Listing 10-2, the script ID passed to OSAExecute for the script context is gContext, a global context provided by the application. The constant kOSAModeNull in the next parameter indicates that no mode flags are set for script execution.
Listing 10-2  A routine that compiles and executes source data

PROCEDURE MyDoNewScript;
VAR
  componentName, scriptText, resultText: AEDesc;
  scriptID, resultID: OSAID;
  myOSAErr, ignoreErr: OSAError;
BEGIN
  {get the scripting component's name so you can show }
  { the user which scripting language to use}
  myOSAErr := OSAScriptingComponentName(gScriptingComponent,
    componentName);
  IF myOSAErr = noErr THEN
    BEGIN
      {get the user's script text, then compile it}
      MyGetUserScriptText(componentName, scriptText);
      {to create a new compiled script using the user's script }
      { for the script to be compiled}
      scriptID := kOSANullScript;
      myOSAErr := OSACompile(gScriptingComponent, scriptText,
        kOSAModeNull, scriptID);
      ignoreErr := AEDisposeDesc(scriptText);
    END;
  IF myOSAErr = noErr THEN
    BEGIN
      {execute the script in a global context}
      myOSAErr := OSAExecute(gScriptingComponent, scriptID,
        gContext, kOSAModeNull, resultID);
      ignoreErr := OSADispose(gScriptingComponent, scriptID);
      IF myOSAErr = noErr THEN
        BEGIN
          {convert the script value returned by OSAExecute to } 
          { text that can be displayed to the user}
          myOSAErr := OSADisplay(gScriptingComponent, resultID,
            typeChar, kOSAModeNull, resultText);
          ignoreErr := OSADispose(gScriptingComponent, resultID);
          {show result to user}
          MyShowUserResult(resultText);
        END;
    END;
  IF myOSAErr = errOSAScriptError THEN
    MyGetScriptErrorInfo;
END;
CHAPTER 10

Scripting Components

If script execution is successful, the MyDoNewScript procedure passes the script ID for the resulting script value to the OSADisplay function and calls the MyShowUserResult procedure to display the script value to the user. It also disposes of the script data for the compiled script. If OSAExecute or OSACompile returns the result code errOSAScriptError, the MyDoNewScript procedure calls the MyGetScriptErrorInfo procedure shown in Listing 10-3, which uses the OSAScriptError function to obtain more information about the error.

Whenever a scripting component routine returns the result code errOSAScriptError, you can use OSAScriptError to obtain more information about the error. The second parameter of the OSAScriptError function is a constant that specifies the kind of error information to be returned, and the third parameter is the descriptor type for the descriptor record in which the additional error information will be returned.

The MyGetScriptErrorInfo procedure in Listing 10-3 calls OSAScriptError three times: once to obtain an error number for either a system error or a scripting component error, once to obtain a text description of the error, and once to obtain error-range information. (For more information about specifying descriptor types for OSAScriptError, see page 10-37.) Finally, the MyGetScriptErrorInfo procedure extracts the starting and ending positions of the error range in the source data and calls the application-defined procedure MyIndicateError to display the error information to the user. Note that your application is responsible for disposing of any descriptor records that are created.

You should use the OSACompile and OSAExecute functions as shown in Listing 10-2 if you expect the user to execute the compiled script several times or manipulate it in some other way. If you want to compile and execute a script just one time and don’t need to keep the compiled script in memory after it has been executed, you can use either OSACompileExecute or OSADoScript if these functions are supported by the scripting component you specify.

The OSACompileExecute function takes a component instance, a descriptor record for the source data to be compiled and executed, a context ID, and a modeFlags parameter. It executes the resulting compiled script, disposes of the compiled script, and returns the script ID for the resulting script value.

The OSADoScript function takes a component instance, a descriptor record for source data, a context ID, a text descriptor type, and a modeFlags parameter. It compiles and executes the script, returns a descriptor record for the text that corresponds to the resulting script value, and disposes of both the compiled script and the script value.
CHAPTER 10

Scripting Components

Listing 10-3  A procedure that uses OSAScriptError to get information about an execution error

PROCEDURE MyGetScriptErrorInfo;
TYPE
  OSErrPtr       = ^OSErr;
  OSErrHandle    = ^OSErrPtr;
VAR
  errorMessage: Handle;
  startPos, endPos: Integer;
  desc, recordDesc: AE::Desc;
  actualType: DescType;
  actualSize: Size;
  scriptErr, myErr, ignoreErr: OSErr;
  myOSAErr: OSA::Error;
BEGIN
  myOSAErr := OSAScriptError(gScriptingComponent,
                               kOSA::ErrorNumber, type::ShortInteger,
                               desc);
  scriptErr := OSErrHandle(desc.dataHandle)^^;
  ignoreErr := AEDisposeDesc(desc);
  myOSAErr := OSAScriptError(gScriptingComponent,
                               kOSA::ErrorMessage, type::Char, desc);
  errorMessage := desc.dataHandle;
  myOSAErr := OSAScriptError(gScriptingComponent,
                               kOSA::ErrorRange, type::OSA::ErrorRange,
                               desc);
  myErr := AE::CoerceDesc (desc, type::AERecord, recordDesc);
  ignoreErr := AEDisposeDesc(desc);
  myErr := AE::GetKeyPtr(recordDesc, keySourceStart,
                         type::ShortInteger, actualType,
                         Ptr(#startPos), sizeOf(startPos),
                         actualSize);
  myErr := AE::GetKeyPtr(recordDesc, keySourceEnd,
                         type::ShortInteger, actualType,
                         Ptr(#endPos), sizeOf(endPos),
                         actualSize);
  ignoreErr := AEDisposeDesc(recordDesc);
  MyIndicateError(scriptErr, errorMessage, startPos, endPos);
  {add your own error checking}
END;
### Saving Script Data

After creating a new script (or after modifying a previously saved script), a user may want to save it.

**IMPORTANT**

Your application should usually save scripts as script data rather than source data, so that it can reload and execute the data without compiling it.

Before saving script data, your application can use the `OSAStore` function to obtain a handle to the data. The `OSAStore` function takes four input parameters: a component instance that identifies a connection with a scripting component, a script ID for the script data to be stored, a desired descriptor type for the descriptor record to be returned, and a parameter that contains mode flags for use by individual scripting components. It returns a descriptor record for the script data in the fifth parameter.

The sections that follow describe the storage formats used by `OSAStore` and the resource and file types for script data.

### Storage Formats for Script Data

The descriptor record returned by `OSAStore` can be either a generic storage descriptor record or a component-specific storage descriptor record:

- **A generic storage descriptor record** is a special kind of descriptor record of type `typeOSAGenericStorage` that can be used to store script data created by any scripting component.

- **A component-specific storage descriptor record** is a descriptor record whose descriptor type is the scripting component subtype value for the scripting component that created the script data.

Figure 10-1 illustrates the logical arrangement of a generic storage descriptor record. The descriptor type for a generic storage descriptor record is always `typeOSAGenericStorage`, and the data referred to by the descriptor record’s handle is always followed by a trailer containing the subtype value for the scripting component that created the script data.

![Figure 10-1](image-url)  
A generic storage descriptor record

<table>
<thead>
<tr>
<th>Descriptor type:</th>
<th>typeOSAGenericStorage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data:</td>
<td>Script data</td>
</tr>
<tr>
<td></td>
<td>Trailer</td>
</tr>
</tbody>
</table>
Figure 10-2 illustrates the logical arrangement of a component-specific storage descriptor record. The descriptor type for a component-specific storage descriptor record is the subtype value for the scripting component that created the script data, and the data referred to by the descriptor record’s handle consists of the script data only, with no trailer.

In most cases it is safest to request a handle to script data in the form of a generic storage descriptor record, regardless of the scripting component subtype you pass to the OSAStore function.

If the presence of the trailer in a generic storage descriptor record does not interfere with the script data, that data may be used for a wide variety of purposes. For example, if an application uses script IDs to refer to XCMDs, it can call OSAStore with a desired type of typeOSSGenericStorage. The data for the resulting descriptor record consists of the XCMD data followed by a trailer indicating that the script data was created by a scripting component that executes XCMDs. Because the trailer does not interfere with the use of the data, the data may actually be used as an XCMD. Thus, an application can save XCMDs as script data and load and execute them after it has opened a connection with the generic scripting component.

However, in some cases adding a trailer to script data may interfere with script execution. For example, suppose the data for a generic storage descriptor record consists of sound data. If a scripting component attempts to play the data from beginning to end as sound data, the trailer will interfere with the resulting sound. In this case, an application must open an explicit connection with a scripting component that can play sounds before saving the data, and then call OSAStore with a desired type that consists of the subtype for that scripting component.

Resource and File Types for Script Data

When the OSAStore function returns a descriptor record of the specified type, your application can save the descriptor record’s data as a resource of type ‘scpt’ or write it to the data fork of a document.

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.
Chapter 10

Scripting Components

```plaintext
CONST
  kOSAGenericScriptingComponentSubtype = 'scpt';
  kOSAScriptResourceType = kOSAGenericScriptingComponentSubtype;
  typeOSAGenericStorage = kOSAScriptResourceType;
```

If you want to save script data as a compiled script file or as a script application, save it as a resource of type 'scpt'. The Script Editor application uses resource ID 128, but you can use any valid resource ID. Save the script comment that accompanies the script data as resources of type 'TEXT' and 'styl' with resource ID 1128. (See Figure 7-1 on page 7-6 for an example of a script comment.) Each script file can contain only one script and one script comment. The file type for a compiled script file should be 'osas'.

A script application has the file type 'APPL'. If a script application has the creator signature 'aplt', a user can initiate execution of the script it contains by opening it from the Finder. If a script application has the creator signature 'dplt' and contains a user-defined handler for the Open Documents event, a user can initiate execution of the handler by dragging a document or folder icon over the script application's icon. For more information about the file formats used for script files, see “Script Editors and Script Files” on page 7-6.

Script applications must include a 'SIZE' resource and two 'CODE' resources with resource IDs 0 and 1. These resources should be identical to those in the sample script application files provided by Apple Computer, Inc. (except that you can change the size of the memory partition). The 'CODE' resources contain bootstrap code that instantiates the script application component. The script application component, which is registered with the Component Manager at startup, provides the code that loads the script to be run and passes the resulting script ID to the appropriate component.

When the user opens a script application from the Finder, the Finder sends the script application an Open Application event. If the scripting component that created the script supports OSAExecuteEvent, the script application component passes the Open Application event and the script ID for the script to OSAExecuteEvent. If the scripting component doesn’t support OSAExecuteEvent, the script application component passes the script ID to OSAExecute.

Loading and Executing Script Data

Figure 7-4 on page 7-13 illustrates how an application might execute a script whenever the user presses the Tab key after entering a customer’s name in the “Customer Name” field of an electronic form. To execute a script in response to some user action, your application must be able to load and execute the script data for a compiled script.

This section describes how to load and execute a previously compiled and saved script. The next section, “Modifying and Recompiling a Compiled Script,” describes how to allow a user to modify a compiled script.
CHAPTER 10

Scripting Components

The OSALoad function takes three input parameters: a component instance that identifies a connection with a scripting component; a descriptor record that contains a handle to the script data to be loaded; and a parameter that contains flags for use by individual scripting components. The function returns, in the fourth parameter, a script ID for the script data.

When your application calls OSALoad with a component instance that identifies a connection with the generic scripting component, the generic scripting component in turn uses a connection with the scripting component that created the script data (if that component is registered with the Component Manager on the local computer). If the descriptor record passed to OSALoad is of type typeOSAGenericStorage, the generic scripting component uses the trailer that follows the script data to determine which scripting component to open a connection with. If the descriptor record’s type is the subtype value for some other scripting component, the generic scripting component does not look for a trailer and uses the descriptor type to identify the scripting component.

When your application calls OSALoad with a component instance that identifies a connection to any scripting component other than the generic scripting component, that component can load script data only if it was saved as the data for a descriptor record whose descriptor type matches the scripting component’s subtype. In this case, however, your application easily can take advantage of additional routines and other special capabilities provided by that scripting component.

It is also possible to call OSALoad using the generic scripting component, then use generic scripting component routines to identify the specific component associated with the loaded script. This allows you to use component-specific routines with a script originally loaded by the generic scripting component. For information about how to do this, see “Routines Used by Scripting Components,” which begins on page 10-92.

The OSALoad function returns a script ID for the loaded script data. The generic scripting component always associates the returned script ID with the scripting component that created the script. In this way, it can use a connection with that component again whenever the client application passes the returned script ID to other scripting component routines.

Listing 10-4 shows a procedure that loads and executes a script. The MyLoadAndExecute procedure takes a handle to script data that was previously saved using a generic storage descriptor record, obtains a script ID for the equivalent compiled script, executes the compiled script in the default context, and disposes of both the compiled script and the resulting script value ID. If the OSAExecute function returns a script execution error, MyLoadAndExecute obtains further information about the error and displays it to the user.
Listing 10-4  A routine that loads and executes script data previously saved using a generic storage descriptor record

PROCEDURE MyLoadAndExecute (scriptData: Handle);
VAR
    scriptDesc: AEDesc;
    scriptID, resultID: OSAID;
    scriptText: AEDesc;
    myOSAErr, ignoreErr: OSAError;
BEGIN
    {load the script data}
    scriptDesc.descriptorType := typeOSAGenericStorage;
    scriptDesc.dataHandle := scriptData;
    myOSAErr := OSALoad(gScriptingComponent, scriptDesc,
                          kOSAModeNull, scriptID);
    IF myOSAErr = noErr THEN
        BEGIN
            {execute the resulting compiled script in the default }
            { context}
            myOSAErr := OSAExecute(gScriptingComponent, scriptID,
                                    kOSANullScript, kOSAModeNull,
                                    resultID);
            ignoreErr := OSADispose(gScriptingComponent, scriptID);
            ignoreErr := OSADispose(gScriptingComponent, resultID);
        END;
        IF myOSAErr = errOSAScriptError THEN
            MyGetScriptErrorInfo;
        END;
END;

The OSALoad function in Listing 10-4 takes a component instance, a generic storage descriptor record for the script data to be loaded, and a parameter that contains the mode flags, if any, for loading the script. In this case the constant kOSAModeNull indicates that no mode flags are set. The OSALoad function returns a script ID for the resulting compiled script, which the MyLoadAndExecute procedure then passes to the OSAExecute function.

In addition to a component instance and the script ID for the compiled script to be executed, the OSAExecute function takes a script ID for a context and a parameter that contains the mode flags, if any, for script execution. In Listing 10-4, the script ID passed to OSAExecute for the script context is kOSANullScript, indicating that the scripting component can use its default context to bind any variables. The constant kOSAModeNull in the next parameter indicates that no mode flags are set for script execution.
After disposing of the compiled script and the resulting script value, MyLoadAndExecute checks the result code returned by OSAExecute. If it is errOSAScriptError, MyLoadAndExecute calls the MyGetScriptErrorInfo procedure (see Listing 10-3 on page 10-11), which in turn uses the OSAScriptError function to obtain more information about the error.

You can use the OSALoad and OSAExecute functions as shown in Listing 10-4 if you expect the user to execute the compiled script several times or manipulate it in some other way. If you want to load and execute a script just one time and don’t need to keep the compiled script in memory after it has been executed, you can use OSALoadExecute instead of OSALoad, OSAExecute, and OSADispose. This function takes a component instance, a descriptor record for the script data to be loaded and executed, a context ID, and a modeFlags parameter. The OSALoadExecute function executes the resulting compiled script, disposes of the compiled script, and returns the script ID for the resulting script data.

Modifying and Recompiling a Compiled Script
In addition to loading and executing a previously compiled and saved script as described in the previous section, your application can use the scripting component routines described in this section to decompile a compiled script, display the equivalent source data to users for editing, and recompile the source data after editing is completed. For example, if a user wants to change the script shown in Figure 7-4 on page 7-13 so that it refers to some other database or looks up other information in addition to the customer’s address, the forms application can use scripting component routines to display the compiled script to the user and recompile it after the user has modified it.

You can use the OSAGetSource function to obtain the source data for a compiled script. The OSAGetSource function takes a component instance, a script ID for the compiled script, and the desired type of the resulting descriptor record. If you specify a component instance that identifies a connection with the generic scripting component, you can use OSAGetSource to get the source data for any compiled script created by a scripting component that is registered with the Component Manager on the local computer. If you specify a component instance that identifies an explicit connection with a scripting component, you can use OSAGetSource only to get the source data for scripts that were compiled by that scripting component.

The MyEditGenericScript procedure in Listing 10-5 shows how you can use the OSAGetSource function with a component instance that identifies a connection to the generic scripting component. The MyEditGenericScript function gets the source data for a compiled script, allows the user to edit it, and recompiles the script so the original script ID refers to the recompiled script data.
CHAPTER 10

Scripting Components

Listing 10-5  A routine that displays a compiled script for editing and recompiles it

PROCEDURE MyEditGenericScript (scriptId: OSAID);
VAR
    scriptText:    AEDesc;
    myOSAErr:      OSAError;
    ignoreErr:     OSErr;
BEGIN
    {first get the source data}
    myOSAErr := OSAGetSource(gScriptingComponent, scriptId,
                               typeChar, scriptText);
    {call the application's primitive text editor}
    MyEditText(scriptText.dataHandle);
    {now compile the edited script data in scriptText using }
    { the scripting component that originally created it; }
    { passing the original script ID to OSACompile causes }
    { OSACompile to replace the original script with the new one}
    myOSAErr := OSACompile(gScriptingComponent, scriptText,
                            kOSAModeNull, scriptId);
    ignoreErr := AEDisposeDesc(scriptText);
    IF myOSAErr = errOSAScriptError THEN
        MyGetScriptErrorInfo;
END;

After obtaining the source data for the script, the MyEditGenericScript procedure calls the MyEditText function, which displays the application's own primitive text editor and allows the user to edit the source data. After the user has finished editing the script, MyEditGenericScript passes the edited text and the script ID for the original compiled script to OSACompile function, which updates the script ID so that it refers to the modified and recompiled script. The kOSAModeNull constant passed in the third parameter of OSACompile indicates that no mode flags are specified for compilation.

If the OSACompile function returns errOSAScriptError, the MyEditGenericScript procedure calls the MyGetScriptErrorInfo procedure shown in Listing 10-3 on page 10-11 to obtain information about the error.

After script data has changed as shown in Listing 10-5, your application should save the modified script data. Listing 10-6 shows how this could be done from a function that loads script data, calls the MyEditGenericScript procedure shown in Listing 10-5 to modify and recompile the script, then saves the modified script data.
CHAPTER 10

Scripting Components

Listing 10-6  A function that loads and modifies script data, then saves it using a generic storage descriptor record

FUNCTION MyLoadAndModifyScriptData (resourceID: Integer) : OSAError;

VAR
    scriptDesc: AEDesc;
    storageDescRec: AEDesc;
    scriptID: OSAID;
    myOSAErr: OSAError;
    ignoreErr: OSErr;
    myHndl: Handle;
BEGIN
    scriptDesc.descriptorType := typeOSAGenericStorage;
    scriptDesc.dataHandle := GetResource(kOSAScriptResourceType, resourceID);
    myOSAErr := OSALoad(gGenericScriptingComponent, scriptDesc, kOSAModeNull, scriptID);
    MyEditGenericScript (scriptId);
    myOSAErr := OSAStore(gScriptingComponent, scriptID, typeOSAGenericStorage, kOSAModeNull, storageDescRec);
    MyWriteResource(storageDescRec.dataHandle, resourceID);
    ignoreErr := AEDisposeDesc(scriptDesc);
    ignoreErr := AEDisposeDesc(storageDescRec);
END;

Using a Script Context to Handle an Apple Event

The preceding sections describe how you can load, compile, modify, and execute scripts under circumstances determined by your application. Your application can use these techniques to associate a script with an Apple event object or application object and execute the script when the user manipulates the object in some way.

Another way to execute a script is to use a script context (called a script object in AppleScript) to handle an Apple event. To do this, your application passes both the event and the script context to OSAExecuteEvent or OSADoEvent. You can also associate script contexts with Apple event objects—that is, objects in your application that can be identified by object specifier records. If an Apple event acts on an object with which a script context is associated, your application attempts to use the script context to handle the Apple event.

For example, Figure 7-7 on page 7-26 shows how you can use a general Apple event handler to provide initial processing for all Apple events received by your application. Listing 10-7 shows an example of such a handler.
You install a general Apple event handler like the one in Listing 10-7 in your application’s special handler dispatch table using the constant keyPreDispatch:

```
myErr := AEInstallSpecialHandler(keyPreDispatch,
                               @MyGeneralAppleEventHandler,
                               FALSE);
```

When it receives an Apple event, the MyGeneralAppleEventHandler function in Listing 10-7 first extracts the event’s direct parameter. It then calls another application-defined function, MyGetAttachedScript, which checks whether the direct parameter contains an object specifier record, calls AEResolve to locate the corresponding Apple event object, and returns a script ID for any script context attached to that object.

If a script context is associated with the object, MyGeneralAppleEventHandler passes the script context’s script ID and the Apple event to the OSADoEvent function. Otherwise, MyGeneralAppleEventHandler returns errAEEventNotHandled, which 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.

The OSADoEvent function in Listing 10-7 takes a component instance that identifies a connection with the generic scripting component. (If it has not already done so, the generic scripting component in turn opens a connection with the scripting component that created the script context.) In addition to the component instance, the Apple event, and the script ID for the script context, OSADoEvent takes a parameter that indicates no mode flags are set and a VAR parameter that contains any reply Apple event returned as a result of handling the event.

If the scripting component determines that a script context can’t handle the specified event (for example, if an AppleScript script context doesn’t include statements that handle the event), OSADoEvent returns errAEEventNotHandled. If OSADoEvent attempts to use the script context to handle the event, the function returns a reply event that contains either the resulting script value or, if an error occurred, information about the error.

The script context shown in Figure 7-7 contains an AppleScript handler for the Move event. Such handlers exist only as AppleScript statements in a script context and do not have corresponding entries in an application’s Apple event dispatch table. However, a handler in a script context can modify or override the actions performed by an application’s standard Apple event handlers installed in its Apple event dispatch table. The next section, “Supplying a Resume Dispatch Function,” describes how this works.
CHAPTER 10

Scripting Components

Listing 10-7  A general Apple event handler that uses the OSADoEvent function

FUNCTION MyGeneralAppleEventHandler (event: AppleEvent;
    reply: AppleEvent;
    refcon: LongInt): OSErr;

VAR dp, resultDesc: AEDesc;
    scriptID: OSAID;
    myErr, ignoreErr: OSErr;
    myOSAErr: OSAError;
BEGIN
    {get the direct parameter}
    myErr := AEGetParamDesc(event, keyDirectObject, typeWildCard,
        dp);
    {get script ID for script context attached to object }
    { specified in direct parameter}
    IF MyGetAttachedScript(dp, scriptID) THEN
        {execute the handler in the script context handler and, if }
        { necessary, the default Apple event handler}
        myOSAErr := OSADoEvent(gScriptingComponent, event,
            scriptID, kOSAModeNull, reply)
    ELSE
        myOSAErr := errAEEventNotHandled;
        ignoreErr := AEDisposeDesc(dp);
        MyGeneralAppleEventHandler := OSErr(myOSAErr);
    END;
END;

For more information about OSADoEvent, OSAExecuteEvent, and other routines
related to the use of script contexts to handle Apple events, see page 10-71.

Supplying a Resume Dispatch Function

Every scripting component calls a resume dispatch function during script execution if
the script contains the equivalent of an AppleScript continue statement within an
event handler. (See Figure 7-7 on page 7-26 for an example.) The resume dispatch
function dispatches the event specified by the script directly to the application’s standard
handler for that event.
Thus, if the script context passed to `OSADoEvent` in Listing 10-7 specifies that the event passed in the `event` parameter should be continued—that is, handled by the application’s standard Apple event handler for that event—the scripting component passes the event to the resume dispatch function currently set for that instance of the scripting component. The resume dispatch function attempts to redispactch the event to the handler installed in the application’s Apple event dispatch table for that event. If the call to the resume dispatch function is successful, execution of the script proceeds from the point at which the resume dispatch function was called. If the call to the resume dispatch function is not successful, `OSADoEvent` returns `errAEEventNotHandled` in the `keyErrorNumber` parameter of the reply event. (Other routines that execute scripts, such as `OSAExecute` or `OSAExecuteEvent`, return `errOSAScriptError` in this situation, and a subsequent call to `OSAScriptError` with `kOSAErrorNumber` in the selector parameter returns `errAEEventNotHandled`.)

Some scripting components may provide routines that allow your application to set or get the pointer to the resume dispatch function used by a specified instance of a scripting component.

```pascal
TYPE AEHandlerProcPtr = EventHandlerProcPtr;

A resume dispatch function takes the same parameters as an Apple event handler.

FUNCTION MyResumeDispatch (theAppleEvent: AppleEvent; reply: AppleEvent; refCon: LongInt):OSErr;

To set the resume dispatch function for a scripting component, call `OSASetResumeDispatchProc`; to get the current dispatch function for a scripting component, call `OSAGetResumeDispatchProc`. If you do not set a resume dispatch function for a scripting component, it uses standard Apple event dispatching to dispatch the event, starting with the special handler dispatch table.

You can install a resume dispatch function using the `OSASetResumeDispatchProc` function. However, if you are using a general handler similar to that in Listing 10-7 on page 10-21 and 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 resume dispatch function and the constant `kOSADontUsePhac` as the reference constant when you call `OSASetResumeDispatchProc`.

```pascal
myErr := OSASetResumeDispatchProc(gScriptingComponent, 
    kOSAUseStandardDispatch, kOSADontUsePhac);
```

This causes the Apple Event Manager to redispactch events that would otherwise be passed to a resume dispatch function using standard Apple event dispatching—except that the Apple Event Manager bypasses your application’s special handler dispatch table and thus won’t call your general Apple event handler recursively.
When a scripting component calls your resume dispatch function, the A5 register is set up for your application, and your application is the current process.

Supplying an Alternative Active Function

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

```pascal
TYPE OSAActiveProcPtr = ProcPtr;

A pointer of type OSAActiveProcPtr points to a MyActiveProc function that takes a reference constant as a parameter.

FUNCTION MyActiveProc(refCon: LongInt): OSErr;
```

If you want your application to get time periodically during script compilation and execution for tasks such as spinning the cursor or checking for system-level errors, you should provide an alternative active function that performs those tasks. To set an alternative active function, call `OSASetActiveProc`; to get the current active function, call `OSAGetActiveProc`.

If you do not 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.

Your alternative active function can in turn call the scripting component’s default active function. To do this, your application can call `OSAGetActiveProc` before calling `OSASetActiveProc` to set the alternative active function, then call the default active function directly when necessary. Some scripting components may also supply building-block routines that your application can use to construct an alternative active function.

Multithreaded applications may need to give time to other threads while one thread is waiting for the scripting component to complete compilation or execution of a script. You can provide an alternative send function and an idle function that allows threads to be switched (see “Alternative Send Functions” on page 10-25). However, the Apple Event Manager calls an idle function only after an Apple event has been sent, whereas a scripting component calls an active function at regular intervals throughout script compilation and execution. Thus, to give time to multiple threads, you may want to provide an alternative active function in addition to an alternative send function and an idle function.

When a scripting component calls your alternative active function, the A5 register is set up for your application, and your application is the current process.
Suppling Alternative Create and Send Functions

Every scripting component calls a *create function* whenever it creates an Apple event during script execution, and a *send function* whenever it sends an Apple event. Scripting components that use Apple events during script compilation, including AppleScript, also call create and send functions during compilation.

Some scripting components may provide routines that allow your application to set or get the pointers to the create and send functions used by that scripting component. If your application does not set alternative send and create functions, the scripting component uses the standard Apple Event Manager functions `AESend` and `AECreateAppleEvent`, which it calls with its own default parameters.

A scripting component that supports the routines you can use to set or get alternative create and send functions has the kOSASupportsAESending bit set in its component description record. For more information about using the Component Manager to find a scripting component that supports specific routines, see “Connecting to a Scripting Component,” which begins on page 10-3.

When a scripting component calls your alternative send or create function, the A5 register is set up for your application, and your application is the current process.

Alternative Create Functions

A scripting component that allows your application to set or get its create function uses a pointer to identify the current create function.

```
TYPE AECreateAppleEventProcPtr = ProcPtr;
```

A pointer of type `AECreateAppleEventProcPtr` points to a `MyAECreateProc` function that takes the same parameters as the `AECreate` function plus a reference constant.

```
FUNCTION MyAECreateProc (theAEEventClass: AEEventClass;
                          theAEEventID: AEEventID;
                          target: AEAddressDesc;
                          returnID: Integer;
                          transactionID: LongInt;
                          VAR result: AppleEvent;
                          refCon: LongInt): OSErr;
```

Your application can use an alternative create function to gain control over the creation and addressing of Apple events. This can be useful, for example, if your application needs to add its own transaction code to the event.
To set an alternative create function, call `OSASetCreateProc`; to get the current create function, call `OSAGetCreateProc`. If you do not set an alternative create function for a scripting component, it uses the standard Apple Event Manager function `AECreateAppleEvent`, which it calls with its own default parameters.

Your alternative create function can in turn call the scripting component’s default create function. To do this, your application can call `OSAGetCreateProc` before calling `OSASetCreateProc` to set the alternative create function, then call the default create function directly when necessary.

### Alternative Send Functions

A scripting component that allows your application to set or get its send function uses a pointer to identify the current send function.

```
TYPE AESendProcPtr = ProcPtr;
```

A pointer of type `AESendProcPtr` points to a `MyAESendProc` function that takes the same parameters as the `AECreate` function plus a reference constant.

```
FUNCTION MyAESendProc (theAppleEvent: AppleEvent;
             VAR reply: AppleEvent;
             sendMode: AESendMode;
             sendPriority: AESendPriority;
             timeOutInTicks: LongInt;
             idleProc: IdleProcPtr;
             filterProc: EventFilterProcPtr;
             refCon: LongInt): OSErr;
```

Your application can use an alternative send function to perform almost any action instead of or in addition to sending Apple events. For example, it can modify Apple events before sending them, save copies of Apple events before sending them, or substitute some other specialized mechanism for sending Apple events.

To set an alternative send function, call `OSASetSendProc`; to get the current send function, call `OSAGetSendProc`. If you do not 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.

Your alternative send function can in turn call the scripting component’s default send function. To do this, your application can call `OSAGetSendProc` before calling `OSASetSendProc` to set the alternative send function, then call the default send function directly when necessary.
After a scripting component successfully calls a send function, the scripting component proceeds with script execution. If a call to a send function is not successful, the scripting component returns `errOSAScriptError`, and a subsequent call to `OSAScriptError` with `kOSAErrorMessage` in the `selector` parameter returns `errAEEventNotHandled`.

Multithreaded applications need to allow other threads to execute while one thread is waiting for the response to an Apple event. You can accomplish this by supplying an idle function for your alternative send function that allows threads to be switched and by setting the `kAEQueueReply` flag in the `sendMode` parameter of the send function. However, if the call to the send function specifies the `kAENoReply` flag, be careful not to override it, because the user may have explicitly requested that no reply be returned or the ‘aete’ resource may indicate that the application cannot reply to that event.

**Note**
The Apple Event Manager calls an idle function only after an Apple event has been sent, whereas a scripting component calls an active function at regular intervals throughout script compilation and execution. Thus, to give time to multiple threads, you may want to provide an alternative active function in addition to an alternative send function and an idle function.

Some scripting components (including the current version of AppleScript) can execute only one script at a time per component instance. For this reason, a multithreaded application must provide a separate component instance for each script that it compiles or executes while it is also compiling or executing other scripts.

You should follow the rules for setting `sendMode` flags described in the chapter “Creating and Sending Apple Events” in this book when you set flags for the `sendMode` parameter of an alternative send function. Keep these additional guidelines in mind:

- If the target application is on the local computer, you can set the `kAECanInteract` and `kAECanSwitchLayer` flags.
- If the target application is on the local computer and the user has requested no reply, set the `kAENoReply`, `kAECanInteract`, and `kAECanSwitchLayer` flags.
- If the target application is on a remote computer, set the `kAENeverInteract` flag and do not set the `kAECanSwitchLayer` flag.

**Recording Scripts**
If you want your application to record Apple events in the form of a compiled script, or if you are writing a script-editing application like Script Editor, you can use the `OSAStartRecording` and `OSAStopRecording` functions to start and stop recording into a specified script ID on a single computer. Both functions take a component instance and a script ID for a compiled script. When your application calls `OSAStartRecording`, the scripting component identified by the component instance
CHAPTER 10

Scripting Components

sends a Start Recording event to your application and installs a Receive Recordable Event handler in your application’s Apple event dispatch table. When your application calls OSAStopRecording, the scripting component removes the handler.

An application acting as a recording process in this manner should not provide a handler for the Start Recording event. Instead, the Apple Event Manager receives the event and responds by sending a Recording On event to all running processes on the local computer. Thereafter, the Apple Event Manager sends copies of subsequent recordable events to the recording process, whose previously installed Receive Recordable Event handler, much like a handler for event class typeWildCard and event ID typeWildCard, handles those recordable events by recording them in the compiled script specified in the call to OSAStartRecording.

Whenever the Receive Recordable Event handler receives a recordable event, the scripting component sends your application a Recorded Text event. The Recorded Text event contains the decompiled source data for the recorded event in the form of styled text.

Recorded Text—append styled text to script editor window

<table>
<thead>
<tr>
<th>Event class</th>
<th>kOSASuite</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event ID</td>
<td>kOSARecordedText</td>
</tr>
<tr>
<td>Required parameter</td>
<td>keyDirectObject</td>
</tr>
<tr>
<td>Keyword:</td>
<td>keyDirectObject</td>
</tr>
<tr>
<td>Descriptor type:</td>
<td>typeStyledText or any other text descriptor type</td>
</tr>
<tr>
<td>Data:</td>
<td>Decompiled source data for recorded event</td>
</tr>
<tr>
<td>Description:</td>
<td>Sent by a scripting component to a recording process for each event recorded after a call to OSAStartRecording</td>
</tr>
</tbody>
</table>

If your want your application to display the source data for recorded events as they are recorded, you must provide a handler for the Recorded Text event.

For more information about the Receive Recordable Event handler and Apple event recording, see “How Apple Event Recording Works,” which begins on page 9-35.

Writing a Scripting Component

It is possible to create scripting components that execute a variety of scripts, including scripts that can be “run” in some sense but do not consist of statements in a scripting language. For example, script data can consist of an XCMD or even sound data that the appropriate scripting component can trigger or play back when it executes the script (see “Storage Formats for Script Data,” which begins on page 10-12).
Chapter 10

Scripting Components

If you are developing a scripting component, you should read the instructions for creating components in the chapter “Component Manager” in Inside Macintosh: More Macintosh Toolbox. Every scripting component should also

- Provide a component name in the scripting component’s component resource that will make sense when displayed to users.
- Support the standard scripting component routines described in “Required Scripting Component Routines,” which begins on page 10-30.
- Support some, all, or none of the optional scripting component routines, as appropriate for the tasks to be performed by the scripting component. These routines are described in “Optional Scripting Component Routines,” which begins on page 10-46.
- Use the three OSA routines OSAGetStorageType, OSAAddStorageType, and OSARemoveStorageType to inspect, add, or remove the trailers appended to script data in generic storage descriptor records. These routines are described in “Manipulating Trailers for Generic Storage Descriptor Records,” which begins on page 10-92.
- Send the Get AETE event when necessary. This event is described in “Handling the Get AETE Event,” which begins on page 8-23.

Scripting Components Reference

This section describes the standard scripting component data structures and routines your application can use to manipulate and execute scripts. This section also describes additional routines provided by the AppleScript scripting component and three routines called by scripting components.

The first section, “Data Structures,” describes the principal data types used by scripting component routines. “Required Scripting Component Routines,” which begins on page 10-30, describes the standard scripting component routines that all scripting components must support. “Optional Scripting Component Routines,” which begins on page 10-46, describes additional standard scripting component routines that scripting components are not required to support.

Your application can use the Component Manager to find a scripting component that supports specific optional routines or to determine whether a particular scripting component supports a specific group of routines. For information about how to do this, see “Connecting to a Scripting Component,” which begins on page 10-3.

“AppleScript Component Routines,” which begins on page 10-80, describes additional routines supported by the AppleScript component. “Generic Scripting Component Routines” which begins on page 10-84, describes routines you can use to get instances of specific components and perform other useful tasks when you are using multiple scripting components. “Routines Used by Scripting Components,” which begins on page 10-92, describes three routines that all scripting components can use to manipulate trailers for generic storage descriptor records.
Data Structures

This section describes the principal data structures and Component Manager type codes used by the standard scripting component routines. Data structures used by individual routines are described with the appropriate routines in the sections that follow.

The Component Manager type code for components that support the standard scripting component routines is 'osa', and the subtype code for the generic scripting component is 'scpt'.

```plaintext
CONST
  kOSACOMPONENTTYPE = 'osa';
  kOSAGENERICSPECIALCOMPONENTTYPE = 'scpt';
```

Because all results returned by the Component Manager are of type ComponentResult (a long integer), scripting components also define this type for result codes.

```plaintext
TYPE
  OSAERROR = ComponentResult;
```

Scripting components keep track of script data in memory by means of script IDs of type OSAID.

```plaintext
TYPE OSAID = LongInt;
```

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.

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 OSAReport, the scripting component creates a new compiled script for editing or recording.

```plaintext
CONST kOSANullScript = 0;
```
CHAPTER 10

Scripting Components

Required Scripting Component Routines

This section describes the routines that all scripting components must support. Your application can use these routines to save and load script data, execute and dispose of scripts, get script information, and manipulate the active function. “Optional Scripting Component Routines,” which begins on page 10-46, describes additional routines your application can use with scripting components that support them.

Saving and Loading Script Data

The OSAStore function takes a script ID and returns a copy of the corresponding script data in the form of a storage descriptor record. You can then save the script data as a resource or write it to the data fork of a document. The OSALoad function takes script data in a storage descriptor record and returns a script ID.

OSAStore

You can use the OSAStore function to get a handle to script data in the form of a storage descriptor record.

FUNCTION OSAStore(scriptingComponent: ComponentInstance; scriptID: OSAID; desiredType: DescType; modeFlags: LongInt; VAR resultingScriptData: AEDesc): OSAError;

scriptingComponent
  A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

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
  The resulting descriptor record.
CHAPTER 10

Scripting Components

DESCRIPTION

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.

CONST

kOSAGenericScriptingComponentSubtype = 'scpt';
kOSAScriptResourceType = kOSAGenericScriptingComponentSubtype;
typeOSAGenericStorage = kOSAScriptResourceType;

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751 Invalid script ID</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752 Desired type not supported by this scripting component</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For more information about storage formats for script data, see “Saving Script Data” on
page 10-12.

For an example of the use of OSAStore, see Listing 10-6 on page 10-19.
SCRIPTING COMPONENTS

CHAPTER 10

OSALoad

You can use the OSALoad function to load script data.

FUNCTION OSALoad(scriptingComponent: ComponentInstance;
                   scriptData: AEDesc;
                   modeFlags: LongInt;
                   VAR resultingScriptID: OSAID): OSAError;

scriptingComponent
    A component instance created by a prior call to the Component Manager
    function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptData
    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
    The returned script ID for the compiled script.

DESCRIPTION

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 OSASave again to
obtain a handle to the modified script data and save it.
CHAPTER 10

Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0     No error</td>
</tr>
<tr>
<td>errOSACorruptData</td>
<td>-1702 Corrupt data</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752 Script data not for this scripting component</td>
</tr>
<tr>
<td>errOSADataFormatObsolete</td>
<td>-1758 Data format is obsolete</td>
</tr>
<tr>
<td>errOSADataFormatTooNew</td>
<td>-1759 Data format is too new</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For more information about the way scripting components interpret script data, see “Saving Script Data” on page 10-12.

For examples of the use of OSAload, see Listing 10-4 on page 10-16 and Listing 10-6 on page 10-19.

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 described in the preceding section or the optional OSAcompile function described on page 10-48 to obtain a script ID.

The OSAExecute function takes a script ID for a compiled script or script context and returns a script ID for a script value. The OSDisplay function converts a script value to text that your application can later display to the user. If the OSAExecute function returns errOSAScriptError, you can use the OSAScriptError function to get more information about the error.

When your application no longer needs the script data associated with a specific script ID, you can use the OSADispose function to release the memory the script data occupies.

OSAExecute

You can use the OSAExecute function to execute a compiled script or a script context.

FUNCTION OSAExecute(scriptingComponent: ComponentInstance; compiledScriptID: OSAID; contextID: OSAID; modeFlags: LongInt; VAR resultingScriptValueID: OSAID): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
CHAPTER 10

Scripting Components

compiledScriptID
The script ID for the compiled script to be 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.

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
The script ID for the script value returned.

DESCRIPTION

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 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 these flags to the modeFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds kAENeverInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds kAECanInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds kAEAlwaysInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCantSwitchLayer</td>
<td>Prevents use of kAECanSwitchLayer in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
<tr>
<td>kOSAModeDontReconnect</td>
<td>Adds kAEReconnect to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDoRecord</td>
<td>Prevents use of kAEDontRecord in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
</tbody>
</table>
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`.

### RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750  General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751 Invalid script ID</td>
</tr>
<tr>
<td>errOSAScriptError</td>
<td>-1753 Error occurred during execution</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

### SEE ALSO

For information about the `OSAGetSource` and `OSACoerceToDesc` functions, see page 10-51 and page 10-54, respectively.

For examples of the use of the `OSAExecute` function, see Listing 10-2 on page 10-9 and Listing 10-4 on page 10-16.

For more information about resume dispatch functions, see “Supplying a Resume Dispatch Function,” which begins on page 10-21, and the description of a resume dispatch function on page 10-97.

### OSADisplay

You can use the `OSADisplay` function to convert a script value to text. Your application can then use its own routines to display this text to the user.

```pascal
FUNCTION OSADisplay(scriptingComponent: ComponentInstance; scriptValueID: OSAID; desiredType: DescType; modeFlags: LongInt; VAR resultingText: AEDesc): OSAError;
```

- **scriptingComponent**: A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).
- **scriptValueID**: The script ID for the script value to coerce.
- **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 The resulting descriptor record.

DESCRIPTION

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 all the standard text descriptor types defined in the Apple Event Registry: Standard Suites, plus any special types supported by the scripting component.

Unlike OSAGetSource, 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.

SPECIAL CONSIDERATIONS

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSACantCoerce</td>
<td>−1700 Desired type not supported by scripting component</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>−1750 General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>−1751 Invalid script ID</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For descriptions of the OSAGetSource and OSACoerceToDesc functions, see page 10-51 and page 10-54, respectively.

For an example of the use of OSADisplay, see Listing 10-2 on page 10-9.
You can use the OSAScriptError function to get information about errors that occur during script execution.

FUNCTION OSAScriptError(scriptingComponent: ComponentInstance; selector: OSType; desiredType: DescType; VAR resultingErrorDescription: AEDesc) : OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

selector
A value that determines what OSAScriptError returns. The value can be one of these constants:

CONST
kOSAErrorNumber = 'errn';
kOSAErrorMessage = 'errs';
kOSAErrorBriefMessage = 'errb';
kOSAErrorApp = 'erap';
kOSAErrorPartialResult = 'ptlr';
kOSAErrorOffendingObject = 'erob';
kOSAErrorRange = 'erng';

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
The resulting descriptor record.

DESCRIPTION
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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Information returned in resultingErrorDescription parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAErrorNumber</td>
<td>Error number for either system error or scripting component error. The value of desiredType must be typeShortInteger.</td>
</tr>
<tr>
<td>kOSAErrorMessage</td>
<td>Error message associated with error number, including both the name of the application and a description of the error. This constant is sufficient for simple error reporting. The value of desiredType must be typeChar or another text descriptor type.</td>
</tr>
<tr>
<td>kOSAErrorBriefMessage</td>
<td>Brief error message associated with error number, excluding the name of the application, any partial result, and the offending object. The value of desiredType must be typeChar or another text descriptor type.</td>
</tr>
<tr>
<td>kOSAErrorApp</td>
<td>Either the name or the process serial number (PSN) of the application that received the error, if it was the result of sending an Apple event. The value of desiredType must be typeProcessSerialNumber (for the PSN) or a text descriptor type such as typeChar (for the name).</td>
</tr>
<tr>
<td>kOSAErrorPartialResult</td>
<td>Partial result returned after a call to AESend that failed. This consists of a reply parameter that contains some but not all of the information requested. The value of desiredType must be typeBest (for the best type) or typeWildCard (for the default type).</td>
</tr>
<tr>
<td>kOSAErrorOffendingObject</td>
<td>An object specifier record for the object that caused the error. The value of desiredType must be typeObjectSpecifier, typeBest, or typeWildCard. For some scripting components, including AppleScript, these three values are equivalent.</td>
</tr>
<tr>
<td>kOSAErrorRange</td>
<td>Range of source data in which error occurred. The value of desiredType must be typeOSAErrorRange.</td>
</tr>
</tbody>
</table>

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.
CONST typeOSAErrorRange = 'erng';

A descriptor record of type typeOSAErrorRange is an AE record that consists of two descriptor records of typeShortInteger specified by these keywords:

CONST
   keyOSASourceStart = 'srcs';  {start of error range}
   keyOSASourceEnd = 'srce';   {end of error range}

If the value of the selector parameter is kOSAErrorNumber, scripting components may return, in the resultingErrorDescription parameter, one of these general error codes:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errOSACantCoerce</td>
<td>-1700 Same as errAECoercionFail; can’t coerce data to requested descriptor type</td>
</tr>
<tr>
<td>errOSACantAccess</td>
<td>-1728 Same as errAENoSuchObject; runtime error in resolution of object specifier record</td>
</tr>
<tr>
<td>errOSAGeneralError</td>
<td>-2700 General runtime error</td>
</tr>
<tr>
<td>errOSADivideByZero</td>
<td>-2701 Attempt to divide by zero</td>
</tr>
<tr>
<td>errOSANumericOverflow</td>
<td>-2702 Numeric overflow</td>
</tr>
<tr>
<td>errOSACantLaunch</td>
<td>-2703 Can’t launch specified file because it isn’t an application</td>
</tr>
<tr>
<td>errOSAAppNotHighLevelEventAware</td>
<td>-2704 Doesn’t respond to Apple events</td>
</tr>
<tr>
<td>errOSACorruptTerminology</td>
<td>-2705 The application has a corrupted ’aete’ resource</td>
</tr>
<tr>
<td>errOSAStackOverflow</td>
<td>-2706 Stack overflow</td>
</tr>
<tr>
<td>errOSAInternalTableOverflow</td>
<td>-2707 Internal table overflow</td>
</tr>
<tr>
<td>errASDataBlockTooLarge</td>
<td>-2708 Attempt to create a value larger than the allowable size</td>
</tr>
<tr>
<td>errOSATypeError</td>
<td>-1703 Same as errAEWrongDataType; wrong descriptor type</td>
</tr>
<tr>
<td>errOSAMessageNotUnderstood</td>
<td>-1708 Same as errAEEventNotHandled; event not handled or message not understood</td>
</tr>
<tr>
<td>errOSAUndefinedMessage</td>
<td>-1717 Same as errAEHandlerNotFound; handler not found for message</td>
</tr>
<tr>
<td>errOSAIllegalIndex</td>
<td>-1728 Same as errAEIllegalIndex; not a valid index</td>
</tr>
<tr>
<td>errOSAIllegalRange</td>
<td>-2720 Same as errAEImpossibleRange; range of specified objects not possible</td>
</tr>
<tr>
<td>errOSASyntaxError</td>
<td>-2740 General syntax error</td>
</tr>
<tr>
<td>errOSASyntaxTypeError</td>
<td>-2741 Syntax error; parser expected one type but found another</td>
</tr>
<tr>
<td>errOSATokenTooLong</td>
<td>-2742 Identifier too long</td>
</tr>
</tbody>
</table>

continued
Scripting Components

Although scripting components are not required to support these error codes, their use simplifies error handling for applications that run scripts created by multiple components.

If the value of the selector parameter is kOSAErrorNumber, the AppleScript component may return, in the resultingErrorDescription parameter, one of these error codes:

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>errOSAMissingParameter -1701</td>
<td>Same as errAEDescNotFound; descriptor record not found</td>
</tr>
<tr>
<td>errOSAParameterMismatch -1721</td>
<td>Same as errAEWrongNumberOfArgs; wrong number of arguments</td>
</tr>
<tr>
<td>errOSADuplicateParameter -2750</td>
<td>Parameter specified more than once</td>
</tr>
<tr>
<td>errOSADuplicateProperty -2751</td>
<td>Property specified more than once</td>
</tr>
<tr>
<td>errOSADuplicateHandler -2752</td>
<td>Handler defined more than once</td>
</tr>
<tr>
<td>errOSAUndefinedVariable -2753</td>
<td>Undefined variable</td>
</tr>
<tr>
<td>errOSAInconsistentDeclarations -2754</td>
<td>Inconsistent declarations</td>
</tr>
<tr>
<td>errOSAControlFlowError -2755</td>
<td>Control flow error</td>
</tr>
</tbody>
</table>

SPECIAL CONSIDERATIONS

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr 0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSACantCoerce -1700</td>
<td>Desired type not supported by scripting component</td>
</tr>
<tr>
<td>errOSASystemError -1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>errOSABadSelector -1754</td>
<td>Selector value not supported by scripting component</td>
</tr>
<tr>
<td>badComponentInstance $80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of OSAScriptError, see Listing 10-3 on page 10-11.
**OSADispose**

You can use the OSADispose function to reclaim the memory occupied by script data.

```pascal
FUNCTION OSADispose(scriptingComponent: ComponentInstance; scriptID: OSAID): OSAError;
```

**DESCRIPTION**

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.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750  General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751   Invalid script ID</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

### Setting and Getting Script Information

The OSASetScriptInfo function sets various kinds of information about script data, and the OSAGetScriptInfo function returns information about script data. The kind of information these functions set or get depends on constants you pass to the functions.

**OSASetScriptInfo**

You can use OSASetScriptInfo to set information about script data according to the value you pass in the selector parameter.

```pascal
FUNCTION OSASetScriptInfo(scriptingComponent: ComponentInstance; scriptID: OSAID; selector: OSType; value: LongInt): OSAError;
```
CHAPTER 10

Scripting Components

scriptingComponent
A component instance created by a prior call to the Component Manager
function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptID The script ID for the script data whose information is to be set.

selector A value that determines what kind of information OSASetScriptInfo
sets. All scripting components can accept this value:

CONST kOSAScriptIsModified = 'modi';

The kOSAScriptIsModified constant indicates that the count of
changes since the script data was loaded or created should be set to the
value in the value parameter. The AppleScript component provides
limited support for this constant.

value The value to set.

DESCRIPTION

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 how many times the script data has
been modified since it was created or passed to OSALoad. Some scripting components
may provide additional constants.

SPECIAL CONSIDERATIONS

Although you can specify kOSAScriptIsModified when you are using the
AppleScript component without generating an error, the current version of AppleScript
doesn’t actually set a value for the count of changes since the script data was loaded or
created. For more information, see the description of OSAGetScriptInfo that follows.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr 0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSASystemError –1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID –1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>errOSABadSelector –1754</td>
<td>Selector value not supported by scripting component</td>
</tr>
<tr>
<td>badComponentInstance $80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>
OSAGetScriptInfo

You can use OSAGetScriptInfo to obtain information about script data according to the value you pass in the selector parameter.

FUNCTION OSAGetScriptInfo(scriptingComponent: ComponentInstance; scriptID: OSAID; selector: OSType; VAR result: LongInt): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptID
The script ID for the script data about which to obtain information.

selector
A value that determines what kind of information OSAGetScriptInfo returns. The value can be one of these constants:

CONST kOSAScriptIsModified = 'modi';
kOSAScriptIsTypeCompiledScript = 'cscr';
kOSAScriptIsTypeScriptValue = 'valu';
kOSAScriptIsTypeScriptContext = 'cntx';
kOSAScriptBestType = 'best';
kOSACanGetSource = 'gsrc';

The AppleScript component provides limited support for the constant kOSAScriptIsModified (see page 10-44). In addition to the standard constants, the AppleScript component also supports this constant:

CONST kASHasOpenHandler = 'hsod';

result
The requested information, which you can coerce to the appropriate descriptor type for the value specified in the selector parameter.

DESCRIPTION
The OSAGetScriptInfo function returns various results according to the value you pass in the selector parameter.

<table>
<thead>
<tr>
<th>Value of selector parameter</th>
<th>Information returned in the result parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAScriptIsModified</td>
<td>Long integer that indicates the number of times the script data has been modified since it was passed to OSALoad.</td>
</tr>
<tr>
<td>kOSAScriptIsTypeCompiledScript</td>
<td>Boolean value that indicates whether or not the script data is a compiled script.</td>
</tr>
</tbody>
</table>

continued
The AppleScript component also provides this constant for use in the selector parameter.

**Value of selector parameter** | **Information returned in the result parameter**
--- | ---
`kOSAScriptIsTypeScriptValue` | Boolean value that indicates whether or not the script data is a script value.
`kOSAScriptIsTypeScriptContext` | Boolean value that indicates whether or not the script data is a script context.
`kOSAScriptBestType` | A descriptor type that you can pass to `OSACoerceToDesc`.
`kOSACanGetSource` | Boolean value that indicates whether the script data can be successfully passed to `OSAGetSource`.

SPECIAL CONSIDERATIONS

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.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>noErr</code></td>
<td>0</td>
</tr>
<tr>
<td><code>errOSASystemError</code></td>
<td>-1750</td>
</tr>
<tr>
<td><code>errOSAInvalidID</code></td>
<td>-1751</td>
</tr>
<tr>
<td><code>errOSABadSelector</code></td>
<td>-1754</td>
</tr>
<tr>
<td><code>badComponentInstance</code></td>
<td>$80008001</td>
</tr>
</tbody>
</table>
Manipulating the Active Function

The OSASetActiveProc and OSAGetActiveProc functions allow your application to set or to get 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.

The functions described in this section use the following type for pointers to active functions:

```c
TYPE OSAActiveProcPtr = ProcPtr;
```

For more information about active functions, see “Supplying an Alternative Active Function” on page 10-23.

### OSASetActiveProc

You can use the OSASetActiveProc routine to set the active function that a scripting component calls periodically while executing a script.

```c
FUNCTION OSASetActiveProc(scriptingComponent: ComponentInstance; activeProc: OSAActiveProcPtr; refCon: LongInt): OSAError;
```

- **scriptingComponent**: A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
- **activeProc**: A pointer to the active function to set. If the value of this parameter is NIL, 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.

### RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
</tr>
</tbody>
</table>
CHAPTER 10

Scripting Components

OSAGetActiveProc

You can use the OSAGetActiveProc function to get a pointer to the active function that a scripting component is currently using.

FUNCTION OSAGetActiveProc(scriptingComponent: ComponentInstance; VAR activeProc: OSAActiveProcPtr; VAR refCon: LongInt): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

activeProc
The OSAGetActiveProc function returns in this parameter a pointer to the active function currently set for the specified scripting component.

refCon
The OSAGetActiveProc function returns in this parameter the reference constant associated with the active function for the specified scripting component.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

Optional Scripting Component Routines

This section describes eight groups of optional routines that scripting components can support. Your application can use the Component Manager to find a scripting component that supports a specific group of routines or to determine whether a particular scripting component supports a specific group of routines.

To specify one or more groups of routines for the Component Manager, use the following constants to set the equivalent bits in the componentFlags field of a component description record:

CONST

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSASupportsCompiling</td>
<td>$0002</td>
</tr>
<tr>
<td>kOSASupportsGetSource</td>
<td>$0004</td>
</tr>
<tr>
<td>kOSASupportsAECoercion</td>
<td>$0008</td>
</tr>
<tr>
<td>kOSASupportsAESending</td>
<td>$0010</td>
</tr>
<tr>
<td>kOSASupportsRecording</td>
<td>$0020</td>
</tr>
<tr>
<td>kOSASupportsConvenience</td>
<td>$0040</td>
</tr>
<tr>
<td>kOSASupportsDialects</td>
<td>$0080</td>
</tr>
<tr>
<td>kOSASupportsEventHandling</td>
<td>$0100</td>
</tr>
</tbody>
</table>
CHAPTER 10

Scripting Components

Each of these flags identifies one of the groups of routines that are described in the sections that follow. For information about using these constants to locate scripting components that support specific groups of optional routines, see “Connecting to a Scripting Component,” which begins on page 10-3.

Compiling Scripts

Scripting components can provide three optional routines that get the name of a scripting component, compile a script, and update a script ID.

To obtain the name of a scripting component in a form that you can coerce to text, you can use the OSAScriptingComponentName function. The OSACompile function compiles source data and returns a script ID, and the OSACopyID function updates the script data associated with one script ID with the script data associated with another script ID.

A scripting component that supports the routines in this section has the kOSASupportsCompiling bit set in the componentFlags field of its component description record.

OSAScriptingComponentName

You can use the OSAScriptingComponentName function to get the name of a scripting component.

FUNCTION OSAScriptingComponentName
    (scriptingComponent: ComponentInstance;
        VAR resultingScriptingComponentName: AEDesc): OSAError;

    scriptingComponent
        A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

    resultingScriptingComponentName
        The name of the scripting component; or, if the component is the generic scripting component, the name of the default scripting component.

DESCRIPTION

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.
CHAPTER 10
Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of OSAScriptingComponentName, see Listing 10-2 on page 10-9.

OSACompile

You can use the OSACompile function to compile the source data for a script and obtain a script ID for a compiled script or a script context.

FUNCTION OSACompile (scriptingComponent: ComponentInstance; sourceData: AEDesc; modeFlags: LongInt; VAR previousAndResultingScriptID: OSAID) : OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

sourceData
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 the description that follows.

previousAndResultingScriptID
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.

DESCRIPTION

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 kOSACompileIntoContext flag, and the source data should include appropriate handlers.
CHAPTER 10

Scripting Components

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.

You can control the way a compiled script is executed by adding any of these flags to the modeFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModePreventGetSource</td>
<td>Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to OSAGetSource.</td>
</tr>
<tr>
<td>kOSACompileIntoContext</td>
<td>The OSACompile function returns a script context instead of a compiled script.</td>
</tr>
<tr>
<td>kOSAModeAugmentContext</td>
<td>Script data associated with script ID passed in previousAndResultingCompiledScriptID is augmented rather than replaced with the new compiled script. Specifying this flag automatically invokes the kOSAModeCompileIntoContext mode flag. If you redefine variables, handlers, and so on that were previously defined in the script context, the new definitions will replace the old ones.</td>
</tr>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds kAENeverInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds kAECanInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds kAEAlwaysInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDontReconnect</td>
<td>Adds kAEDontReconnect to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCantSwitchLayer</td>
<td>Prevents use of kAECanSwitchLayer in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
<tr>
<td>kOSAModeDoRecord</td>
<td>Prevents use of kAEDontRecord in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
</tbody>
</table>

SPECIAL CONSIDERATIONS

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.
Scripting Components

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>noErr, No error</td>
</tr>
<tr>
<td>-1700</td>
<td>errOSACantCoerce, Source data incompatible with scripting component</td>
</tr>
<tr>
<td>-1750</td>
<td>errOSASystemError, General scripting system error</td>
</tr>
<tr>
<td>-1751</td>
<td>errOSAInvalidID, Invalid script ID</td>
</tr>
<tr>
<td>-1753</td>
<td>errOSAScriptError, Source data invalid (syntax error)</td>
</tr>
<tr>
<td>$8008001</td>
<td>badComponentInstance, Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of OSACompile to update an existing script ID, see Listing 10-5 on page 10-18. For an example of the use of OSACompile to obtain a new script ID, see Listing 10-2 on page 10-9.

For more information about the default scripting component associated with any instance of the generic scripting component, see “Generic Scripting Component Routines,” which begins on page 10-84.

OSACopyID

You can use the OSACopyID function to update script data after editing or recording and to perform undo or revert operations on script data.

FUNCTION OSACopyID(scriptingComponent: ComponentInstance; fromID: OSAID; VAR toID: OSAID): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

fromID
The script ID for script data that you want to be associated with the script ID in the toID parameter.

toID
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.

DESCRIPTION

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.

10-50  Scripting Components Reference
CHAPTER 10

Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>errOSAINvalidID</td>
<td>-1751 Invalid script ID</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

Getting Source Data

The OSAGetSource function returns the source data that corresponds to the script data identified by a script ID. The source data it returns can in turn be passed to OSACompile.

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

OSAGetSource

You can use the OSAGetSource function to decompile the script data identified by a script ID and obtain the equivalent source data.

FUNCTION OSAGetSource(scriptingComponent: ComponentInstance; scriptID: OSAID; desiredType: DescType; VAR resultingSourceData: AEDesc): OSAError;

scriptingComponent
  A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

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).

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

resultingSourceData
  The resulting descriptor record.

DESCRIPTION

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751</td>
</tr>
<tr>
<td>errOSASourceNotAvailable</td>
<td>-1756</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
</tr>
</tbody>
</table>

SEE ALSO

For an example of the use of OSAGetSource, see Listing 10-5 on page 10-18.

Coercing Script Values

Scripting components can provide support for two optional routines, OSACoerceFromDesc and OSACoerceToDesc, which coerce data in a descriptor record to a script value and coerce a script value to data in a descriptor record, respectively.

A scripting component that supports the routines in this section has the kOSA.SupportsAECoercion bit set in the componentFlags field of its component description record.

OSACoerceFromDesc

You can use the OSACoerceFromDesc function to obtain the script ID for a script value that corresponds to the data in a descriptor record.

FUNCTION OSACoerceFromDesc

    (scriptingComponent: ComponentInstance;
     scriptData: AEDesc; modeFlags: LongInt;
     VAR resultingScriptValueID: OSAID): OSAError;

10-52 Scripting Components Reference
### Scripting Components

**scriptComponent**  
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).

**scriptData**  
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 the description that follows.

**resultingScriptValueID**  
The resulting script ID for a script value.

### DESCRIPTION

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:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kOSAModePreventGetSource</code></td>
<td>Compiled script consists of only the minimum script data required to run the script. It will cause an error if passed to <code>OSAGetSource</code>.</td>
</tr>
<tr>
<td><code>kOSACompileIntoContext</code></td>
<td>The <code>OSACoerceFromDesc</code> function returns a script context instead of a compiled script.</td>
</tr>
<tr>
<td><code>kOSAModeNeverInteract</code></td>
<td>Adds <code>kAENeverInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td><code>kOSAModeCanInteract</code></td>
<td>Adds <code>kAECanInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td><code>kOSAModeAlwaysInteract</code></td>
<td>Adds <code>kAEAlwaysInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td><code>kOSAModeDontReconnect</code></td>
<td>Adds <code>kAEDontReconnect</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td><code>kOSAModeCantSwitchLayer</code></td>
<td>Prevents use of <code>kAECanSwitchLayer</code> in <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
<tr>
<td><code>kOSAModeDoRecord</code></td>
<td>Prevents use of <code>kAE DontRecord</code> in <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
</tbody>
</table>
CHAPTER 10

Scripting Components

SPECIAL CONSIDERATIONS

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For more information about the default scripting component associated with any instance of the generic scripting component, see “Generic Scripting Component Routines,” which begins on page 10-84.

OSACoerceToDesc

You can use the OSACoerceToDesc function to coerce a script value to a descriptor record of a desired descriptor type.

FUNCTION OSACoerceToDesc(scriptingComponent: ComponentInstance; scriptValueID: OSAID; desiredType: DescType; modeFlags: LongInt; VAR result: AEDesc): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptValueID
The script ID for the script value to coerce.

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
The resulting descriptor record.

DESCRIPTION

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 defined in the *Apple Event Registry: Standard Suites*, plus any special types supported by the scripting component.

**SPECIAL CONSIDERATIONS**

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`.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750  General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751     Invalid script ID</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$8008001 Invalid component instance</td>
</tr>
</tbody>
</table>

**SEE ALSO**

For a description of `OSADisplay`, see page 10-35.

**Manipulating the Create and Send Functions**

Some scripting components provide routines 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 `AECreateAppleEvent` and `AESend` functions with default parameters.

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`.

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`.

The functions described in this section use the following types for pointers to the create and send functions:

```c
typedef void (*AESendProcPtr) (ProcPtr);
typedef void (*AECreateAppleEventProcPtr) (ProcPtr);
```

For more information about create and send functions, see “Supplying Alternative Create and Send Functions,” which begins on page 10-24.
Scripting Components

Scripting components that support manipulation of the create and send functions also support the OSASetDefaultTarget function, which allows you to set the default application to which Apple events are sent.

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**

You can use the OSASetCreateProc function to specify a create function that a scripting component should use instead of the Apple Event Manager’s AECreateAppleEvent function when creating Apple events.

FUNCTION OSASetCreateProc(scriptingComponent: ComponentInstance; createProc: AECreateAppleEventProcPtr; refCon: LongInt): OSAError;

- **scriptingComponent**
  A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
- **createProc**
  A pointer to the create function to set.
- **refCon**
  A reference constant.

**RESULT CODES**

- noErr  0 No error
- errOSASystemError  -1750 General scripting system error
- badComponentInstance  $80008001 Invalid component instance

**OSAGetCreateProc**

You can use the OSAGetCreateProc function to get a pointer to the create function that a scripting component is currently using to create Apple events.

FUNCTION OSAGetCreateProc(scriptingComponent: ComponentInstance; VAR createProc: AECreateAppleEventProcPtr; VAR refCon: LongInt): OSAError;

- **scriptingComponent**
  A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
CHAPTER 10

Scripting Components

createProc

The OSAGetCreateProc function returns, in this parameter, a pointer to the create function currently set for the specified scripting component.

refCon

The OSAGetCreateProc function returns, in this parameter, the reference constant associated with the create function for the specified scripting component.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

OSASetSendProc

You can use the OSASetSendProc function to specify a send function that a scripting component should use instead of the Apple Event Manager’s AESend function when sending Apple events.

FUNCTION OSASetSendProc(scriptingComponent: ComponentInstance;
                        sendProc: AESendProcPtr;
                        refCon: LongInt): OSAError;

scriptingComponent

A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

sendProc

A pointer to the send function to set.

refCon

A reference constant.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

OSAGetSendProc

You can use the OSAGetSendProc function to get a pointer to the send function that a scripting component is currently using.

FUNCTION OSAGetSendProc(scriptingComponent: ComponentInstance;
                         VAR sendProc: AESendProcPtr;
                         VAR refCon: LongInt): OSAError;

Scripting Components Reference
Scripting Components

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

sendProc
The OSAGetSendProc function returns, in this parameter, a pointer to the send function currently set for the specified scripting component.

refCon
The OSAGetSendProc function returns, in this parameter, the reference constant associated with the send function for the specified scripting component.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
</tr>
</tbody>
</table>

OSASetDefaultTarget

You can use the OSASetDefaultTarget function to set the default target application for Apple events.

FUNCTION OSASetDefaultTarget

(oscriptingComponent: ComponentInstance;

target: AEAddressDesc): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

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.

DESCRIPTION

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 routines) as the default target application.
CHAPTER 10

Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

Recording Scripts

The OSAStartRecording function turns on the Apple Event Manager’s recording mechanism and specifies a script in which subsequent recordable Apple events are recorded. The scripting component sends the recording process (for example, a script editor) a Recorded Text event that contains the decompiled equivalent for each recordable event it receives. The script editor can then display the decompiled script in a script editor window if a window for that script is currently open. Recording continues until a call to OSAStopRecording turns recording off.

Script editors use these routines to allow users to control recording. Any application can use these routines to provide its own script-recording interface.

For more information about the Apple event recording mechanism, see the chapter “Recording Apple Events” in this book. For more information about the Recorded Text event, see “Recording Scripts” on page 10-26.

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

You can use the OSAStartRecording routine to turn on Apple event recording and record subsequent Apple events in a compiled script.

FUNCTION OSAStartRecording

(scriptingComponent: ComponentInstance;
VAR compiledScriptToModifyID: OSAID): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

compiledScriptToModifyID
The script ID for the compiled script in which to record.
CHAPTER 10

Scripting Components

DESCRIPTION

The OSAStartRecording routine 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.

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.

SPECIAL CONSIDERATIONS

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

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>-1732</td>
<td>Attempt to turn recording on when it is already on for a recording process</td>
</tr>
<tr>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>-1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>$80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For more information about the default scripting component associated with any instance of the generic scripting component, see “Generic Scripting Component Routines,” which begins on page 10-84.

OSAStopRecording

You can use the OSAStopRecording function to turn off Apple event recording.

FUNCTION OSAStopRecording(scriptingComponent: ComponentInstance; compiledScriptID: OSAID): OSAError;

scriptingComponent

A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
CHAPTER 10

Scripting Components

compiledScriptID
A script ID for the compiled script into which Apple events are being recorded.

DESCRIPTION

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>-1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>$80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

Executing Scripts in One Step

The OSALoadExecute, OSACompileExecute, and OSADoScript functions combine the capabilities of several other scripting component functions so that an application can execute a script in a single 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.

OSALoadExecute

You can use the OSALoadExecute function to load and execute a script in a single step rather than calling OSAload and OSAExecute.

FUNCTION OSALoadExecute (scriptingComponent: ComponentInstance; scriptData: AEDesc; contextID: OSAID; modeFlags: LongInt; VAR resultingScriptValueID: OSAID) : OSAError;
scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptData
The descriptor record identifying the script data to be loaded and executed.

defaultScript
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.

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
The script ID for the script value returned.
CHAPTER 10

Scripting Components

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`.

RESULT CODES

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
</tr>
<tr>
<td>errOSACorruptData</td>
<td>-1702</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752</td>
</tr>
<tr>
<td>errOSAScriptError</td>
<td>-1753</td>
</tr>
<tr>
<td>errOSADATAFormatObsolete</td>
<td>-1758</td>
</tr>
<tr>
<td>errOSADATAFormatTooNew</td>
<td>-1759</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
</tr>
</tbody>
</table>

**OSACompileExecute**

You can use the `OSACompileExecute` routine to compile and execute a script in a single step rather than calling `OSACompile` and `OSAExecute`.

```pascal
FUNCTION OSACompileExecute
    (scriptingComponent: ComponentInstance;
     sourceData: AEDesc;
     contextID: OSAID; modeFlags: LongInt;
     VAR resultingScriptValueID: OSAID): OSAError;
```

- **scriptingComponent**
  A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).

- **sourceData**
  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.

- **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**
  The script ID for the script value returned.
CHAPTER 10

Scripting Components

DESCRIPTION

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.

You can control the way in which the scripting component executes a script by adding any of these flags to the modeFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds kAENeverInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds kAECanInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds kAEAlwaysInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCantSwitchLayer</td>
<td>Prevents use of kAECanSwitchLayer in sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDontReconnect</td>
<td>Adds kAEDontReconnect to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDoRecord</td>
<td>Prevents use of kAERecord in sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
</tbody>
</table>

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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSACantCoerce</td>
<td>−1700 Data could not be coerced to the requested data type</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>−1750 General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>−1751 Invalid script ID</td>
</tr>
<tr>
<td>errOSAScriptError</td>
<td>−1753 Source data invalid (syntax error) or an execution error occurred</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>
OSADoScript

You can use the OSADoScript routine to compile and execute a script and convert the resulting script value to text in a single step rather than calling OSACompile, OSAExecute, and OSADisplay.

FUNCTION OSADoScript (scriptingComponent: ComponentInstance; sourceData: AEDesc; contextID: OSAID; desiredType: DescType; modeFlags: LongInt; VAR resultingText: AEDesc): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

sourceData
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.

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 the description that follows.

resultingText
The resulting descriptor record.

DESCRIPTION
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.
You can control the way in which the scripting component executes the script by adding any of these flags to the `modeFlags` parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds <code>kAENeverInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds <code>kAECanInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds <code>kAEAlwaysInteract</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCantSwitchLayer</td>
<td>Prevents use of <code>kAECanSwitchLayer</code> in <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
<tr>
<td>kOSAModeDontReconnect</td>
<td>Adds <code>kAEDontReconnect</code> to <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDoRecord</td>
<td>Prevents use of <code>kAEDontRecord</code> in <code>sendMode</code> parameter of <code>AESend</code> for events sent when script is executed (the opposite of the Apple Event Manager’s interpretation of the same bit).</td>
</tr>
<tr>
<td>kOSAModeDisplayForHumans</td>
<td>Resulting text is readable by humans only and cannot be recompiled by <code>OSACompile</code>.</td>
</tr>
</tbody>
</table>

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 the result code is `errOSAScriptError`, an error occurred during script execution, and the `resultingText` parameter contains the error message associated with the error. In this case, you can obtain more detailed error information by calling `OSAScriptError`.

### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSACantCoerce</td>
<td>–1700 Data could not be coerced to the requested data type</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>–1750 General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>–1751 Invalid script ID</td>
</tr>
<tr>
<td>errOSAScriptError</td>
<td>–1753 Source data invalid (syntax error) or an execution error occurred</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

### SEE ALSO

For more information about resume dispatch functions, see “Supplying a Resume Dispatch Function,” which begins on page 10-21, and the description of a resume dispatch function on page 10-97.
CHAPTER 10

Scripting Components

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.

The OSASetCurrentDialect function sets the current dialect, and the OSAGetCurrentDialect function gets the dialect code for the current dialect. The OSAAvailableDialectCodeList function returns a list of codes for a scripting component’s dialects. You can pass any of these codes to the OSAGetDialectInfo function to get information about a specific dialect.

Instead of using the OSAAvailableDialectCodeList and OSAGetDialectInfo functions, you can use the OSAAvailableDialects function to get a descriptor list that contains information about all of the currently available dialects for a scripting component. However, it is usually more convenient to get information about just one dialect.

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

You can use the OSASetCurrentDialect function to set the current dialect for a scripting component.

FUNCTION OSASetCurrentDialect
(scriptingComponent: ComponentInstance;
dialectCode: Integer): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

dialectCode
The code for the dialect to be set.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>errOSANoSuchDialect</td>
<td>-1757 Invalid dialect code</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>
OSAGetCurrentDialect

You can use the OSAGetCurrentDialect function to get the dialect code for the dialect currently being used by a scripting component.

FUNCTION OSAGetCurrentDialect
    (scriptingComponent: ComponentInstance;
    VAR resultingDialectCode: Integer): OSAError;

scriptingComponent
    A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

resultingDialectCode
    The OSAGetCurrentDialect function returns, in this parameter, the code for the current dialect of the specified scripting component.

RESULT CODES

| noErr | 0 | No error |
| errOSASystemError | -1750 | General scripting system error |
| errOSANoSuchDialect | -1757 | Invalid dialect code |
| badComponentInstance | $80008001 | Invalid component instance |

OSAAvailableDialectCodeList

You can use the OSAAvailableDialectCodeList function to obtain a descriptor list containing dialect codes for each of a scripting component’s currently available dialects.

FUNCTION OSAAvailableDialectCodeList
    (scriptingComponent: ComponentInstance;
    VAR resultingDialectCodeList: AEDesc): OSAError;

scriptingComponent
    A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

resultingDialectCodeList
    The returned descriptor list.

DESCRIPTION

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.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

**OSAGetDialectInfo**

You can use the `OSAGetDialectInfo` function to get information about a specified dialect provided by a specified scripting component.

```pascal
OSAGetDialectInfo (scriptingComponent: ComponentInstance;
  dialectCode: Integer; selector: OSType;
  VAR resultingDialectInfo: AEDesc): OSAError;
```

- **scriptingComponent**
  A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).

- **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 that follows for a list of the standard constants you can specify in this parameter.

- **resultingDialectInfo**
  A descriptor record containing the requested information. The descriptor record’s descriptor type corresponds to the constant specified in the `selector` parameter.

**DESCRIPTION**

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 following constants for this parameter:
Individual scripting components may allow you to specify additional constants.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>errOSABadSelector</td>
<td>-1754 Invalid selector</td>
</tr>
<tr>
<td>errOSANoSuchDialect</td>
<td>-1757 Invalid dialect code</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

OSAAvailableDialects

You can use the OSAAvailableDialects function to obtain a descriptor list containing information about each of the currently available dialects for a scripting component.

FUNCTION OSAAvailableDialects

```
FUNCTION OSAAvailableDialects
    (scriptingComponent: ComponentInstance;
     VAR resultingDialectInfoList: AEDesc): OSAError;
```

- scriptingComponent: A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).
- resultingDialectInfoList: The returned descriptor list.

DESCRIPTION

Each item in the list returned by OSAAvailableDialects is an AE record of descriptor type typeOSADialectInfo.

CONST typeOSADialectInfo = 'difo';
Each descriptor record in the descriptor list contains, at a minimum, four keyword-specified descriptor records with the following keywords:

```CONST
keyOSADialectName = 'dnam';{used with descriptor record }
    { of any text type, such as }
    { type typeChar}
keyOSADialectCode = 'dcod';{used with descriptor record }
    { of type typeShortInteger}
keyOSADialectLangCode = 'dlcd';{used with descriptor record }
    { of type typeShortInteger}
keyOSADialectScriptCode = 'dscd';{used with descriptor record }
    { of type typeShortInteger}
```

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.

### RESULT CODES

- `noErr` 0 No error
- `errOSASystemError` -1750 General scripting system error
- `badComponentInstance` $80008001 Invalid component instance

### Using Script Contexts to Handle Apple Events

The optional routines 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. (For an example of such a handler, see “Using a Script Context to Handle an Apple Event” on page 10-19.)

The general Apple event handler extracts the event’s direct parameter, obtains a script ID for the script context associated with the object described in the direct parameter, and passes the Apple event and the script ID to either `OSAExecuteEvent` or `OSADoEvent`. The main difference between these two functions is that `OSAExecuteEvent` returns a script ID for the resulting script value, whereas `OSADoEvent` returns a reply Apple event that includes either the resulting script value or information about any errors that occurred.

If the scripting component determines that a script context can’t handle the specified event (for example, if an AppleScript script context doesn’t include statements that handle the event), `OSAExecuteEvent` and `OSADoEvent` return `errAEEventNotHandled`. 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 a
script context passed to OSAExecuteEvent or OSDoEvent can handle the event, the
function attempts to use the script context for that purpose.

Script contexts can in turn pass an event to a resume dispatch function with
a statement that’s equivalent to an AppleScript continue statement. The
OSASetResumeDispatchProc and OSAGetResumeDispatchProc functions allow
your application to set and get pointers to the resume dispatch function used by a
scripting component. These functions use the following type for a pointer to a
resume dispatch function:

TYPE AEHandlerProcPtr = EventHandlerProcPtr;

A resume dispatch function takes the same parameters as an Apple event handler and
dispatches an event to an application’s standard handler for that event.

If you need to create a new, empty script context, you can use the OSAMakeContext
function.

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

You can use the OSASetResumeDispatchProc function to set the resume dispatch
function called by a scripting component during execution of an AppleScript continue
statement or its equivalent.

FUNCTION OSASetResumeDispatchProc

(scriptingComponent: ComponentInstance;
 resumeDispatchProc: AEHandlerProcPtr;
 refCon: LongInt): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager
function OpenDefaultComponent or OpenComponent (see page 10-4).

resumeDispatchProc
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 kOSANoDispatch constant, which tells the Apple Event Manager
  that the processing of the Apple event is complete and that it does not
  need to be dispatched
Scripting Components

refCon A reference constant. Specify kOSADontUsePhac in this parameter and kOSAUseStandardDispatch in the resumeDispatchProc parameter to request standard Apple event dispatching excluding the special handler dispatch table.

DESCRIPTION

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 similar to that in Listing 10-7 on page 10-21 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 general Apple event handler recursively.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>noErr</td>
</tr>
<tr>
<td>-1750</td>
<td>errOSASystemError</td>
</tr>
<tr>
<td>$80008001</td>
<td>badComponentInstance</td>
</tr>
</tbody>
</table>

SEE ALSO

For more information about resume dispatch functions, see “Supplying a Resume Dispatch Function” on page 10-21 and the description of a resume dispatch function on page 10-97.

OSAGetResumeDispatchProc

You can use the OSAGetResumeDispatchProc function to get the resume dispatch function currently being used by a scripting component instance during execution of an AppleScript continue statement or its equivalent.

FUNCTION OSAGetResumeDispatchProc

```pascal
(scriptingComponent: ComponentInstance;
 VAR resumeDispatchProc: AEHandlerProcPtr;
 VAR refCon: LongInt): OSAError;
```
CHAPTER 10

Scripting Components

**scriptingComponent**
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).

**resumeDispatchProc**
The `OSAGetResumeDispatchProc` function returns a pointer to the resume dispatch function for the specified scripting component in this parameter. If no resume dispatch function has been registered, `OSAGetResumeDispatchProc` returns `kOSAUseStandardDispatch` (the default).

**refCon**
The `OSAGetResumeDispatchProc` function returns the reference constant associated with the resume dispatch function in this parameter.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>noErr</code></td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td><code>errOSASystemError</code></td>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td><code>badComponentInstance</code></td>
<td>$80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

**OSAExecuteEvent**

You can use the `OSAExecuteEvent` function to handle an Apple event with the aid of a script context and obtain a script ID for the resulting script value.

```pascal
FUNCTION OSAExecuteEvent(scriptingComponent: ComponentInstance;
                          theAppleEvent: AppleEvent;
                          contextID: OSAID;
                          modeFlags: LongInt;
                          VAR resultingScriptValueID: OSAID):
                          OSAError;
```

**scriptingComponent**
A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).

**theAppleEvent**
The Apple event to be handled.

**contextID**
The script ID for the script context to be used to handle the Apple event.

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

**resultingScriptValueID**
A script ID for the resulting script value.
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 errAEEventNotHandled rather than errOSAScriptError.

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.

You can control the way in which the scripting component executes a script context by adding any of these flags to the modeFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds kAENeverInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds kAECanInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds kAEAlwaysInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCantSwitchLayer</td>
<td>Prevents use of kAECanSwitchLayer in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager's interpretation of the same bit).</td>
</tr>
<tr>
<td>kOSAModeDontReconnect</td>
<td>Adds kAEDontReconnect to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeDoRecord</td>
<td>Prevents use of kAEDontRecord in sendMode parameter of AESend for events sent when script is executed (the opposite of the Apple Event Manager's interpretation of the same bit).</td>
</tr>
</tbody>
</table>

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.

**IMPORTANT**
The `OSAExecuteEvent` function can generate the result code `errAEEventNotHandled` 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 `errAEEventNotHandled`. 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 `errAEEventNotHandled` as the resulting error description.

### RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>-1708</td>
<td>Script context doesn’t contain handler for event</td>
</tr>
<tr>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>-1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>-1753</td>
<td>Error occurred during execution or because of an attempt to pass event to a NIL resume dispatch function</td>
</tr>
<tr>
<td>$80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

### `OSADoEvent`

You can use the `OSADoEvent` function to handle an Apple event with the aid of a script context and obtain a reply event.

**FUNCTION OSADoEvent**(scriptingComponent: ComponentInstance; theAppleEvent: AppleEvent; contextID: OSAID; modeFlags: LongInt; VAR reply: AppleEvent): OSAError;

- **scriptingComponent**: A component instance created by a prior call to the Component Manager function `OpenDefaultComponent` or `OpenComponent` (see page 10-4).
- **theAppleEvent**: The Apple event to be handled.
CHAPTER 10

Scripting Components

contextID  The script ID for the script context to be used to handle the Apple event.
modeFlags  Information used by individual scripting components. To avoid setting any mode flags, specify kOSAModeNull. Other possible mode flags are listed in the description that follows.
reply  The reply Apple event.

DESCRIPTION

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 errAEEventNotHandled rather than errOSAScriptError.

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.

You can control the way in which the scripting component executes a script context by adding any of these flags to the modeFlags parameter:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>kOSAModeNeverInteract</td>
<td>Adds kAENeverInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeCanInteract</td>
<td>Adds kAECanInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
<tr>
<td>kOSAModeAlwaysInteract</td>
<td>Adds kAEAlwaysInteract to sendMode parameter of AESend for events sent when script is executed.</td>
</tr>
</tbody>
</table>

continued
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.

**IMPORTANT**
Like `OSAExecuteEvent`, `OSADoEvent` can generate the result code `errAEEventNotHandled` 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 `errAEEventNotHandled`. 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 `errAEEventNotHandled` in the reply Apple event. ▲

### RESULT CODES

- **noErr**: 0 No error
- **errAEEventNotHandled**: -1708 Script context doesn’t contain handler for event
- **errOSASystemError**: -1750 General scripting system error
- **errOSAInvalidID**: -1751 Invalid script ID
- **badComponentInstance**: $80008001 Invalid component instance

### SEE ALSO

For an example of the use of `OSADoEvent`, see Listing 10-7 on page 10-21.
OSAMakeContext

You can use the OSAMakeContext function to get a script ID for a new script context.

FUNCTION OSAMakeContext(scriptingComponent: ComponentInstance;
contextName: AEDesc;
parentContext: OSAID;
VAR resultingContextID: OSAID): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager
function OpenDefaultComponent or OpenComponent (see page 10-4).

certaintyName
Name of 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
Existing context from which 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 script ID for the resulting script context.

DESCRIPTION
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.

SPECIAL CONSIDERATIONS
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.

RESULT CODES

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSACantCoerce</td>
<td>-1700</td>
<td>Invalid context name</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
<td>Invalid component instance D</td>
</tr>
</tbody>
</table>

SEE ALSO
To compile existing source data into a script context, use OSACompile as described on
page 10-48.
AppleScript Component Routines

The AppleScript component provides routines for initializing the AppleScript component and manipulating the styles used to display AppleScript statements in a script. These routines are used primarily by script editors and other applications that display source data to users.

Initializing AppleScript

Before you call any of the standard scripting component routines, you can call the ASInit function to initialize the AppleScript component with desired application-specific stack and heap sizes. If you don’t call ASInit, the AppleScript component initializes itself using either the values specified in the application’s ‘scsz’ resource or, for those values not provided by the ‘scsz’ resource, default values provided by the AppleScript component.

**ASInit**

You can use the ASInit function to initialize the AppleScript component.

FUNCTION ASInit (scriptingComponent: ComponentInstance; modeFlags: LongInt; minStackSize: LongInt; preferredStackSize: LongInt; maxStackSize: LongInt; minHeapSize: LongInt; preferredHeapSize: LongInt; maxHeapSize: LongInt): OSAError;

- **scriptingComponent**
  A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

- **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 stack.

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

- **maxHeapSize**
  The maximum size for the portion of the application’s heap used by the AppleScript component’s application-specific stack.
CHAPTER 10

Scripting Components

`minHeapSize`
The minimum size for the portion of the application’s heap used by the AppleScript component’s application-specific heap.

`preferredHeapSize`
The preferred size for the portion of the application’s heap used by the AppleScript component’s application-specific heap.

`maxHeapSize`
The maximum size for the portion of the application’s heap used by the AppleScript component’s application-specific heap.

**DESCRIPTION**

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 corresponding default value:

```
CONST
kASDefaultMinStackSize = 1 * 1024;
kASDefaultPreferredStackSize = 4 * 1024;
kASDefaultMaxStackSize = 16 * 1024;
kASDefaultMinHeapSize = 4 * 1024;
kASDefaultPreferredHeapSize = 64 * 1024;
kASDefaultMaxHeapSize = 32 * 1024 * 1024;
```

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 routines 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.
CHAPTER 10

Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750  General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

Getting and Setting Styles for Source Data

The ASGetSourceStyles and ASSetSourceStyles functions allow you to get and set the script format styles currently used by the AppleScript component to display scripts. To obtain a list of style names formatted according to the script format styles currently used by the AppleScript component, use the ASGetSourceStyleNames function.

**ASGetSourceStyles**

You can use the ASGetSourceStyles function to get the script format styles currently used by the AppleScript component to display scripts.

FUNCTION ASGetSourceStyles

```
FUNCTION ASGetSourceStyles
    (scriptingComponent: ComponentInstance;
        VAR resultingSourceStyles: STHandle): OSAError;
```

**DESCRIPTION**

The ASGetSourceStyles function returns a style element array that defines the nine styles used for AppleScript terms.

You can use these index constants to identify individual styles returned in the resultingSourceStyles parameter:

```
CONST
    kASSourceStyleUncompiledText = 0;
    kASSourceStyleNormalText = 1;
    kASSourceStyleLanguageKeyword = 2;
    kASSourceStyleApplicationKeyword = 3;
    kASSourceStyleComment = 4;
    kASSourceStyleLiteral = 5;
```
Scripting Components

kASSourceStyleUserSymbol = 6;
kASSourceStyleObjectSpecifier = 7;
kASNumberOfSourceStyles = 8;

Other AppleScript dialects may define additional styles. When you have finished using the style element array, you must dispose of it.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750 General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For information about the TEStyleTable array, see Inside Macintosh: Text.

ASSetSourceStyles

You can use the ASSetSourceStyles function to set the script format styles used by the AppleScript component to display scripts.

FUNCTION ASSetSourceStyles (scriptingComponent: ComponentInstance; sourceStyles: STHandle): OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

sourceStyles
A handle to a style element array defined by the TextEdit data type TEStyleTable that defines the nine styles used for different kinds of AppleScript terms. The style for each kind of term should be identified according to the index constants listed for ASGetSourceStyles on page 10-82.

DESCRIPTION

The ASSetSourceStyles function sets the script format styles used to display scripts. If you pass a NIL 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.
CHAPTER 10

Scripting Components

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr 0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSASystemError -1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance $80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

SEE ALSO

For information about the TEStyleTable array, see *Inside Macintosh: Text*.

**ASGetSourceStyleNames**

You can use the ASGetSourceStyleNames function to obtain a list of style names that are each formatted according to the script format styles currently used by the AppleScript component.

FUNCTION ASGetSourceStyleNames

(scriptingComponent: ComponentInstance;
 modeFlags: LongInt;
 VAR resultingSourceStyleNameList: AEDescList)
: OSAError;

scriptingComponent
A component instance created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

modeFlags Reserved for future use. Set to kOSAModeNull.

resultingSourceStyleNameList
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 for ASGetSourceStyles on page 10-82.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr 0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSASystemError -1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>badComponentInstance $80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>

Generic Scripting Component Routines

To manipulate and execute scripts written in different scripting languages, your application can either open a connection with each corresponding scripting component individually or open a single connection with the generic scripting component. For information about how to connect with scripting components, see “Connecting to a Scripting Component,” which begins on page 10-3.
If you open a connection with the generic scripting component, it in turn attempts to open connections dynamically with the appropriate scripting component for each script that it executes or manipulates. To provide this capability, the generic scripting component must be able to determine which scripting component created any script ID passed as a parameter to a standard scripting component routine. Because different scripting components may end up using the same script ID to refer to different scripts, the generic scripting component uses its own \textit{generic script IDs}. The generic scripting component translates generic scripting IDs into the corresponding component-specific script IDs and vice versa when necessary.

A generic script ID is a script ID of type \texttt{GenericID}.

\begin{verbatim}
TYPE GenericID = OSAID;
\end{verbatim}

You don’t need to know in detail how the generic scripting component keeps track of script IDs. However, you should be aware that the script IDs to which your application refers when it uses the generic scripting component are not the same as the script IDs used by scripting components that actually manipulate and execute scripts.

If you are writing a script editor or recorder, you must pass the existing script ID to \texttt{OSACompile} or \texttt{OSAStartRecording} when you are recompiling or recording into an existing script. This ensures that the script is recompiled or recorded using the same scripting component that originally created the script. If instead you pass \texttt{kOSANullScript} to these routines, the new script is compiled or recorded using the default scripting component. Each instance of the generic scripting component has its own default scripting component. The section “Getting and Setting the Default Scripting Component,” which follows, describes routines provided by the generic scripting component that allow you to get and set the default scripting component.

The generic scripting component supports the standard scripting component routines. However, most scripting components also support their own 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 facilitates 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. The section “Using Component-Specific Routines,” which begins on page 10-87, describes the generic scripting component routines that allow you to perform these tasks.

Some generic scripting component routines take or return a component subtype of type \texttt{ScriptingComponentSelector}.

\begin{verbatim}
TYPE ScriptingComponentSelector = OSType;
\end{verbatim}

You can use subtype codes of this type to identify specific scripting components.
CHAPTER 10

Scripting Components

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. The OSAGetDefaultScriptingComponent and OSASetDefaultScriptingComponent functions allow you to get and set the default scripting component.

**OSAGetDefaultScriptingComponent**

You can use the OSAGetDefaultScriptingComponent function to get the subtype code for the default scripting component associated with an instance of the generic scripting component.

**FUNCTION OSAGetDefaultScriptingComponent**

(genericScriptingComponent: ComponentInstance;
VAR scriptingSubType: ScriptingComponentSelector)
: OSAError;

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptingSubType
The function returns, in this parameter, the subtype code for the default scripting component associated with the instance of the generic scripting component specified in the genericScriptingComponent parameter.

**DESCRIPTION**

The OSAGetDefaultScriptingComponent function returns the subtype code for the default scripting component. This is the scripting component that will be used by OSAStartRecording, 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.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr 0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSACantOpenComponent -1762</td>
<td>Can’t connect to scripting component</td>
</tr>
<tr>
<td>badComponentInstance $80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>
OSASetDefaultScriptingComponent

You can use the OSASetDefaultScriptingComponent function to set the default scripting component associated with an instance of the generic scripting component.

**FUNCTION**

```plaintext
FUNCTION OSASetDefaultScriptingComponent
  (genericScriptingComponent: ComponentInstance;
   scriptingSubType: ScriptingComponentSelector)
  : OSAError;
```

- **genericScriptingComponent**
  A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

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

**DESCRIPTION**

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.

**RESULT CODES**

- **noErr**
  0 No error
- **errOSACantOpenComponent**
  -1762 Specified component subtype hasn’t been registered
- **badComponentInstance**
  $80008001 Invalid component instance

**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.
CHAPTER 10

Scripting Components

If you want to identify the scripting component that created a storage descriptor record but don’t want to load the script, use the OSAGetScriptingComponentFromStored function. When you need to use a specific scripting component, the OSAGetScriptingComponent function allows you to get a component instance for that scripting component.

The OSAGenericToRealID and OSARealToGenericID functions allow you to convert between generic script IDs and component-specific script IDs.

OSAGetScriptingComponentFromStored

You can use the OSAGetScriptingComponentFromStored routine to get the subtype code for a scripting component that created a storage descriptor record.

FUNCTION OSAGetScriptingComponentFromStored
(genericScriptingComponent: ComponentInstance;
scriptData: AEDesc;
VAR scriptingSubType: ScriptingComponentSelector)
: OSAError;

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptData
Either a generic storage descriptor record or a component-specific storage descriptor record.

scriptingSubType
The function returns, in this parameter, a subtype code identifying the scripting component that created the descriptor record specified by the scriptData parameter.

DESCRIPTION

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.
RESULT CODES

- noErr: 0 (No error)
- errOSACantOpenComponent: -1762 (Can't connect to scripting component)
- badComponentInstance: $80008001 (Invalid component instance)

OSAGetScriptingComponent

You can use the OSAGetScriptingComponent function to get the instance of a scripting component for a specified subtype.

FUNCTION OSAGetScriptingComponent
(genericScriptingComponent: ComponentInstance;
scriptingSubType: ScriptingComponentSelector;
VAR scriptingInstance: ComponentInstance)
: OSAError;

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

scriptingSubType
A subtype code for a scripting component.

scriptingInstance
The function returns, in this parameter, a component instance for the scripting component identified by the scriptingSubType parameter.

DESCRIPTION

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 OSAGetDefaultComponent 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`.

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSACantOpenComponent</td>
<td>-1762 Can't connect to scripting component</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001 Invalid component instance</td>
</tr>
</tbody>
</table>

**SEEALSO**

For descriptions of the `OSAGetDefaultScriptingComponent` and `OSAScriptingComponentName` functions, see page 10-86 and page 10-47, respectively.

**OSAGenericToRealID**

You can use the `OSAGenericToRealID` function to convert a generic script ID to the corresponding component-specific script ID.

```swift
FUNCTION OSAGenericToRealID
    (genericScriptingComponent: ComponentInstance;
     VAR theScriptID: OSAID;
     VAR theExactComponent: ComponentInstance)
    : OSAError;

genericScriptingComponent
    A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

theScriptID
    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.

theExactComponent
    The `OSAGenericToRealID` function returns, in this parameter, the component instance that created the script ID returned in the `theScriptID` parameter.
```
DESCRIPTION

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 OSGenericToRealID.

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 OSGenericToRealID function returns the equivalent component-specific script ID and the component instance that created that script ID. The OSGenericToRealID 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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0 No error</td>
</tr>
<tr>
<td>errOSACantOpenComponent</td>
<td>-1762  Can’t connect to scripting component</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001  Invalid component instance</td>
</tr>
</tbody>
</table>

**OSARealToGenericID**

You can use the OSARealToGenericID function to convert a component-specific script ID to the corresponding generic script ID.

FUNCTION OSARealToGenericID

(genericScriptingComponent: ComponentInstance;
VAR theScriptID: OSAID;
theExactComponent: ComponentInstance)
: OSAError;

genericScriptingComponent
A component instance for the generic scripting component, created by a prior call to the Component Manager function OpenDefaultComponent or OpenComponent (see page 10-4).

theScriptID
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.

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

### RESULT CODES

- **noErr**: 0  No error
- **errOSAComponentMismatch**: -1761  The `theScriptID` and `theExactComponent` parameters are for two different scripting components
- **errOSACantOpenComponent**: -1762  Can’t connect to scripting component
- **badComponentInstance**: $80008001  Invalid component instance

### Routines Used by Scripting Components

Scripting components can call three routines to manipulate the trailers for generic storage descriptor records. “Writing a Scripting Component” on page 10-27 provides general guidelines for writing a scripting component.

### 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.

For more information about generic storage descriptor records, see “Saving Script Data,” which begins on page 10-12.
CHAPTER 10

Scripting Components

**OSAGetStorageType**

You can use the OSAGetStorageType function to retrieve the scripting component subtype from the script trailer appended to the script data in a generic storage descriptor record.

```pascal
FUNCTION OSAGetStorageType(scriptData: Handle;
    VAR type: DescType): OSErr;
```

*scriptData*  
A handle to the script data.

*type*  
The descriptor type specified in the script data trailer.

**DESCRIPTION**

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

**RESULT CODES**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0  No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750  General scripting system error</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752  Bad storage type</td>
</tr>
</tbody>
</table>

**OSAAddStorageType**

You can use the OSAAddStorageType routine to add a trailer to the script data in a generic storage descriptor record.

```pascal
FUNCTION OSAAddStorageType(scriptData: Handle;
    type: DescType): OSErr;
```

*scriptData*  
A handle to the script data.

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

**DESCRIPTION**

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0  No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750   General scripting system error</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752   Bad storage type</td>
</tr>
</tbody>
</table>

**OSARemoveStorageType**

You can use the OSARemoveStorageType routine to remove a trailer from the script data in a generic storage descriptor record.

**FUNCTION** OSARemoveStorageType (scriptData: Handle): OSErr;

- **scriptData**: A handle to the script data.

**DESCRIPTION**

The OSARemoveStorageType routine 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.

RESULT CODES

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0  No error</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750   General scripting system error</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752   Bad storage type</td>
</tr>
</tbody>
</table>

**Application-Defined Routines**

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.

This section provides the syntax declarations for the active, send, create, and resume dispatch functions. When a scripting component calls any of these functions, the A5 register is set up for your application, and your application is the current process.
Your application can provide an alternative active function that performs periodic tasks during script compilation such as checking for Command-period, spinning the cursor, and checking for system-level errors.

FUNCTION MyActiveProc (refCon: LongInt): OSErr;

refCon A reference constant.

DESCRIPTION

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.

SEE ALSO

For descriptions of the scripting component routines you can use to set or get the pointer to a scripting component’s active function, see “Manipulating the Active Function” on page 10-45.

For a discussion of the role of an active function, see “Supplying an Alternative Active Function” on page 10-23.

Your application can provide an alternative create function to gain control over the creation and addressing of Apple events. This 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 AECreateAppleEvent function plus a reference constant.

FUNCTION MyAECreateProc (theAEEventClass: AEEventClass;
theAEEventID: AEEventID;
target: AEAddressDesc;
returnID: Integer;
transactionID: LongInt;
VAR result: AppleEvent;
refCon: LongInt): OSErr;
**CHAPTER 10**

Scripting Components

**DESCRIPTION**

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.

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

**SEE ALSO**

For descriptions of the scripting component routines you can use to set or get the pointer to a scripting component’s create function, see “Manipulating the Create and Send Functions,” which begins on page 10-55.

For information about create functions, see “Alternative Create Functions” on page 10-24.

For a description of the parameters for the `AECreateAppleEvent` function, see “Creating Apple Events,” which begins on page 5-26.

---

**MyAESendProc**

Your application can provide an alternative send function that performs almost any action instead of or in addition to sending Apple events. 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.

```pascal
```

**DESCRIPTION**

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.
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.

SEE ALSO
For descriptions of the scripting component routines you can use to set or get the pointer to a scripting component’s send function, see “Manipulating the Create and Send Functions,” which begins on page 10-55.
For more information about send functions, see “Alternative Send Functions” on page 10-25.
For a description of the parameters for the `AESend` function, see “Sending Apple Events,” which begins on page 5-38.

MyResumeDispatch

Your application can provide a resume dispatch function that a scripting component calls during script execution to dispatch Apple events directly to an application’s default handler for an Apple event. A resume dispatch function takes the same parameters as an Apple event handler.

FUNCTION MyResumeDispatch (theAppleEvent: AppleEvent; reply: AppleEvent; refCon: LongInt) : OSErr;

theAppleEvent  The Apple event to be dispatched.
reply          The default reply Apple event provided by the Apple Event Manager.
refCon         The reference constant stored in the Apple event dispatch table for the Apple event.

DESCRIPTION
If a script specifies that the Apple event should be passed to an application’s standard handler for that event (for example, with an AppleScript `continue` statement), the scripting component executing the script passes the event to the resume dispatch function currently being used by the scripting component. The resume dispatch function should dispatch the event directly to the application’s standard handler for that event. If you use script contexts to handle Apple events, you may need to provide a resume dispatch function.
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 use the `OSASetResumeDispatchProc` routine to specify that the Apple Event Manager should use standard Apple event dispatching instead of a resume dispatch function.

**SEE ALSO**

For a description of the `OSASetResumeDispatchProc` function, see page 10-72.

For a discussion of the use of script contexts to handle Apple events, see “Using a Script Context to Handle an Apple Event” on page 10-19.
CHAPTER 10

Scripting Components

Summary of Scripting Components

Pascal Summary

Constants

CONST
{Component Manager type and subtype codes}
kOSAComponentType = 'osa';
kOSAGenericScriptingComponentSubtype = 'scpt';

{null script ID passed to OSAExecute, OSAGetSource, or OSAStrartRecording}
kOSANullScript = 0;   {empty script}

{mode flag that indicates a routine's default mode settings are to be }
{ used}
kOSAModeNull = 0;
kOSANullMode = 0;

{mode flag used with OSAStore to store a script context without storing }
{ its parent context}
kOSAModeDontStoreParent = $00004000;

{mode flag used with OSAStore, OSALoad, or OSACompile to request }
{ minimum script data}
kOSAModePreventGetSource = $00000001;

{mode flags used with OSACompile, OSAExecute, OSALoalExecute, }
{ OSAExecuteEvent, and OSADoEvent}

{these mode flags cause the scripting component to set the corresponding }
{ sendMode flags when it sends the event}
kOSAModeNeverInteract = kAENeverInteract;
kOSAModeCanInteract = kAECanInteract;
kOSAModeAlwaysInteract = kAEAlwaysInteract;
kOSAModeDontReconnect = kAEDontReconnect;
Summary of Scripting Components

{this mode flag causes the scripting component not to set the }
{ kAECanSwitchLayer sendMode flag when it sends the event}
kOSAModeCantSwitchLayer = $00000040;

{this mode flag causes the scripting component not to set the }
{ kAEDontRecord sendMode flag when it sends the event}
kOSAModeDoRecord = $00001000;

{mode flags used with OSACompile}

{this mode flag causes OSACompile to compile the source data as a script }
{ context}
kOSAModeCompileIntoContext = $00000002;

{this mode flag causes OSACompile to augment the script data for a }
{ script context rather than replacing it}
kOSAModeAugmentContext = $00000004;

{mode flags used with OSADisplay or OSADoScript to indicate that output }
{ needs to be readable by humans only and does not have to be recompiled }
{ by OSACompile}
kOSAModeDisplayForHumans = $00000008;

{suite and event code for the Recorded Text event}
kOSASuite = 'ascr';
kOSAReducedText = 'recd';

{resource type for stored script data}
kOSAScriptResourceType = kOSAGenericScriptingComponentSubtype;

typeOSAGenericStorage = kOSAScriptResourceType;

{descriptor types and error range keywords for OSAScriptError}
kOSAErrorNumber = 'errn'; {returns error number}
kOSAErrorMessage = 'errs'; {returns error message}
kOSAErrorBriefMessage = 'errb'; {returns brief error }
{ message}
kOSAErrorApp = 'erap'; {returns PSN or name of }
{ errant application}
kOSAErrorPartialResult = 'ptlr'; {returns partial }
CHAPTER 10

Scripting Components

kOSAErrorOffendingObject = 'erob'; {returns info about offending object, if any}
kOSAErrorRange = 'erng'; {returns error range}
typeOSAErrorRange = 'erng'; {descriptor type for error range}
keyOSASourceStart = 'srcs'; {start of error range}
keyOSASourceEnd = 'srce'; {end of error range}

{if selector parameter of kOSAScriptError is kOSAErrorMessageNumber, scripting components may return any of these error codes}

dynamic errors
errOSACantCoerce = errAECoercionFail;
errOSACantAccess = errAENoSuchObject;
errOSAGeneralError = -2700;
errOSADivideByZero = -2701;
errOSANumericOverflow = -2702;
errOSACantLaunch = -2703;
errOSAAppNotHighLevelEventAware = -2704;
errOSACorruptTerminology = -2705;
errOSAStackOverflow = -2706;
errOSAInternalTableOverflow = -2707;
errOSADataBlockTooLarge = -2708;

{component-specific dynamic script errors: -2720 through -2739}

static errors
errTypeError = errAEWrongDataType;
errOSAMessageNotUnderstood = errAEEventNotHandled;
errOSAUndefinedMessage = errAEHandlerNotFound;
errOSAIllegalIndex = errAEImpossibleRange;
errOSASyntaxError = -2740;
errOSASyntaxTypeError = -2741;
errOSATokenTooLong = -2742;
errOSAMissingParameter = errAEDescNotFound;
errOSAParameterMismatch = errAEWrongNumberOfArgs;
errOSADuplicateParameter = -2750;
errOSADuplicateProperty = -2751;
errOSADuplicateHandler = -2752;
errOSAUndefinedVariable = -2753;
Summary of Scripting Components

errOSAInconsistentDeclarations = -2754;
errOSAControlFlowError = -2755;

{component-specific static script errors: -2760 through -2779}

{dialect-specific script errors: -2780 through -2799}

{descriptor type for each item in list returned by OSAAvailableDialects}
typeOSADialectInfo = 'difo';

{keywords for descriptor record of descriptor type typeOSADialectInfo; }
{ these can also be used in selector parameter of OSAGetDialectInfo}
keyOSADialectName = 'dnam'; {used with descriptor }
{ record of any text }
{ type, such as typeChar}
keyOSADialectCode = 'dcod'; {used with descriptor }
{ record of type }
{ typeShortInteger}
keyOSADialectLangCode = 'dlcd'; {used with descriptor }
{ record of type }
{ typeShortInteger}
keyOSADialectScriptCode = 'dscd'; {used with descriptor }
{ record of type }
{ typeShortInteger}

{constants for use with OSASetResumeDispatchProc}
kOSAUseStandardDispatch = kAEUseStandardDispatch;
kOSANoDispatch = kAENoDispatch;
kOSADontUsePhac = $0001;

{selectors for use with OSAGetScriptInfo}
kOSAScriptIsModified = 'modi';
kOSAScriptIsTypeCompiledScript = 'cscr';
kOSAScriptIsTypeScriptValue = 'valu';
kOSAScriptIsTypeScriptContext = 'cntx';
kOSAScriptBestType = 'best';
kOSACanGetSource = 'gsrc';

{OSA component flags}
kOSASupportsCompiling = $0002;
kOSASupportsGetSource = $0004;
kOSASupportsAECoercion = $0008;
kOSASupportsAESending = $0010;
kOSASupportsRecording = $0020;
kOSASupportsConvenience = $0040;
kOSASupportsDialects = $0080;
kOSASupportsEventHandling = $0100;

{component selectors}

{basic scripting}
kOSASelectLoad = $0001;
kOSASelectStore = $0002;
kOSASelectExecute = $0003;
kOSASelectDisplay = $0004;
kOSASelectScriptError = $0005;
kOSASelectDispose = $0006;
kOSASelectSetScriptInfo = $0007;
kOSASelectGetScriptInfo = $0008;
kOSASelectSetActiveProc = $0009;
kOSASelectSetActiveProc = $000A;

{compiling}
kOSASelectScriptingComponentName = $0102;
kOSASelectCompile = $0103;
kOSASelectCopyID = $0104;

{getting source data}
kOSASelectGetSource = $0201;

{coercing script values}
kOSASelectCoerceFromDesc = $0301;
kOSASelectCoerceToDesc = $0302;

{manipulating send and create functions}
kOSASelectSetSendProc = $0401;
kOSASelectGetSendProc = $0402;
kOSASelectSetCreateProc = $0403;
kOSASelectGetCreateProc = $0404;
kOSASelectSetDefaultTarget = $0405;

{recording}
kOSASelectStartRecording = $0501;
kOSASelectStopRecording = $0502;
CHAPTER 10

Scripting Components

{convenience}
kOSASelectLoadExecute = $0601;
kOSASelectCompileExecute = $0602;
kOSASelectDoScript = $0603;

{manipulating dialects}
kOSASelectSetCurrentDialect = $0701;
kOSASelectGetCurrentDialect = $0702;
kOSASelectAvailableDialects = $0703;
kOSASelectGetDialectInfo = $0704;
kOSASelectAvailableDialectCodeList = $0705;

{executing Apple event handlers in script contexts}
kOSASelectSetResumeDispatchProc = $0801;
kOSASelectGetResumeDispatchProc = $0802;
kOSASelectExecuteEvent = $0803;
kOSASelectDoEvent = $0804;
kOSASelectMakeContext = $0805;

{scripting-component-specific selectors begin with this value}
kOSASelectComponentSpecificStart = $1001;

{******AppleScript component constants******}
typeAppleScript = 'ascr';

{Component Manager subtype for AppleScript component}
kAppleScriptSubtype = typeAppleScript;

{AppleScript constant for storage descriptor records}
typeASStorage = typeAppleScript;

{AppleScript constant for the selector parameter of OSAGetScriptInfo}
kASHasOpenHandler = 'hsod';

{AppleScript component selectors}
kASSelectInit = $1001;
kASSelectSetSourceStyles = $1002;
kASSelectGetSourceStyles = $1003;
kASSelectGetSourceStyleNames = $1004;

{default initialization parameters for AppleScript}
kASDefaultMinStackSize = 1 * 1024;
kASDefaultPreferredStackSize = 4 * 1024;
kASDefaultMaxStackSize = 16 * 1024;
CHAPTER 10

Scripting Components

kASDefaultMinHeapSize = 4 * 1024;
kASDefaultPreferredHeapSize = 64 * 1024;
kASDefaultMaxHeapSize = 32 * 1024 * 1024;

{AppleScript source style flags}
kASSourceStyleUncompiledText = 0;
kASSourceStyleNormalText = 1;
kASSourceStyleLanguageKeyword = 2;
kASSourceStyleApplicationKeyword = 3;
kASSourceStyleComment = 4;
kASSourceStyleLiteral = 5;
kASSourceStyleUserSymbol = 6;
kASSourceStyleObjectSpecifier = 7;
kASNumberOfSourceStyles = 8;

{if selector parameter of kOSAScriptError is kOSAErrorNumber, }
{ AppleScript component may return any of these error codes}
errASCantConsiderAndIgnore = -2720;
errASCantCompareMoreThan32k = -2721;
errASCantCompareMixedScripts = -2722;
errASTerminologyNestingTooDeep = -2760;
errASInconsistentNames = -2780; {English dialect}

{*******generic scripting component constants*******}

{component version this header file describes}
kGenericComponentVersion = $0100;

{generic scripting component selectors}
kGSSSelectGetDefaultScriptingComponent = $1001;
kGSSSelectSetDefaultScriptingComponent = $1002;
kGSSSelectGetScriptingComponent = $1003;
kGSSSelectGetScriptingComponentFromStored = $1004;
kGSSSelectGenericToRealID = $1005;
kGSSSelectRealToGenericID = $1006;

Data Types

TYPE
OSAID = LongInt; {script ID}
OSAError = ComponentResult; {type for result codes}

Summary of Scripting Components 10-105
CHAPTER 10

Scripting Components

{pointers for application-defined functions}
OSAActiveProcPtr = ProcPtr;
AESendProcPtr = ProcPtr;
AECreatAppleEventProcPtr = ProcPtr;
AEHandlerProcPtr = EventHandlerProcPtr;

{generic scripting component data types}
ScriptingComponentSelector = OSType;
GenericID = OSAID;

Required Scripting Component Routines

**Saving and Loading Script Data**

FUNCTION OSAStore (scriptingComponent: ComponentInstance;
scriptID: OSAID;
desiredType: DescType;
modeFlags: LongInt;
VAR resultingScriptData: AEDesc): OSAError;

FUNCTION OSALoad (scriptingComponent: ComponentInstance;
scriptData: AEDesc;
modeFlags: LongInt;
VAR resultingScriptID: OSAID): OSAError;

**Executing and Disposing of Scripts**

FUNCTION OSAExecute (scriptingComponent: ComponentInstance;
compiledScriptID: OSAID;
contextID: OSAID;
modeFlags: LongInt;
VAR resultingScriptValueID: OSAID): OSAError;

FUNCTION OSADisplay (scriptingComponent: ComponentInstance;
scriptValueID: OSAID;
desiredType: DescType;
modeFlags: LongInt;
VAR resultingText: AEDesc): OSAError;

FUNCTION OSAScriptError (scriptingComponent: ComponentInstance;
selector: OSTYPE;
desiredType: DescType;
VAR resultingErrorDescription: AEDesc): OSAError;

FUNCTION OSADispose (scriptingComponent: ComponentInstance;
scriptID: OSAID): OSAError;
CHAPTER 10

Scripting Components

Summary of Scripting Components

Setting and Getting Script Information
FUNCTION OSASetScriptInfo (scriptingComponent: ComponentInstance; scriptID: OSAID; selector: OSType; value: LongInt): OSAError;
FUNCTION OSAGetScriptInfo (scriptingComponent: ComponentInstance; scriptID: OSAID; selector: OSType; VAR result: LongInt): OSAError;

Manipulating the Active Function
FUNCTION OSASetActiveProc (scriptingComponent: ComponentInstance; activeProc: OSAActiveProcPtr; refCon: LongInt): OSAError;
FUNCTION OSAGetActiveProc (scriptingComponent: ComponentInstance; VAR activeProc: OSAActiveProcPtr; VAR refCon: LongInt): OSAError;

Optional Scripting Component Routines

Compiling Scripts
FUNCTION OSAScriptingComponentName (scriptingComponent: ComponentInstance; VAR resultingScriptingComponentName: AEDesc): OSAError;
FUNCTION OSACompile (scriptingComponent: ComponentInstance; sourceData: AEDesc; modeFlags: LongInt; VAR previousAndResultingScriptID: OSAID): OSAError;
FUNCTION OSACopyID (scriptingComponent: ComponentInstance; fromID: OSAID; VAR toID: OSAID): OSAError;

Getting Source Data
FUNCTION OSAGetSource (scriptingComponent: ComponentInstance; scriptID: OSAID; desiredType: DescType; VAR resultingSourceData: AEDesc): OSAError;

Coercing Script Values
FUNCTION OSACoerceFromDesc (scriptingComponent: ComponentInstance; scriptData: AEDesc; modeFlags: LongInt; VAR resultingScriptValueID: OSAID): OSAError;
CHAPTER 10

Scripting Components

FUNCTION OSACoerceToDesc (scriptingComponent: ComponentInstance;
scriptValueID: OSAID;
desiredType: DescType; modeFlags: LongInt;
VAR result: AEDesc): OSAError;

Manipulating the Create and Send Functions

FUNCTION OSASetCreateProc (scriptingComponent: ComponentInstance;
createProc: AECreateAppleEventProcPtr;
refCon: LongInt): OSAError;

FUNCTION OSAGetCreateProc (scriptingComponent: ComponentInstance;
VAR createProc: AECreateAppleEventProcPtr;
VAR refCon: LongInt): OSAError;

FUNCTION OSASetSendProc (scriptingComponent: ComponentInstance;
sendProc: AESendProcPtr;
refCon: LongInt): OSAError;

FUNCTION OSAGetSendProc (scriptingComponent: ComponentInstance;
VAR sendProc: AESendProcPtr;
VAR refCon: LongInt): OSAError;

FUNCTION OSASetDefaultTarget (scriptingComponent: ComponentInstance;
target: AEAddressDesc): OSAError;

Recording Scripts

FUNCTION OSAStartRecording (scriptingComponent: ComponentInstance;
VAR compiledScriptToModifyID: OSAID): OSAError;

FUNCTION OSAStopRecording (scriptingComponent: ComponentInstance;
compiledScriptID: OSAID): OSAError;

Executing Scripts in One Step

FUNCTION OSALoadExecute (scriptingComponent: ComponentInstance;
scriptData: AEDesc;
contextID: OSAID; modeFlags: LongInt;
VAR resultingScriptValueID: OSAID): OSAError;

FUNCTION OSACompileExecute (scriptingComponent: ComponentInstance;
sourceData: AEDesc;
contextID: OSAID; modeFlags: LongInt;
VAR resultingScriptValueID: OSAID): OSAError;

FUNCTION OSADoScript (scriptingComponent: ComponentInstance;
sourceData: AEDesc;
contextID: OSAID;
desiredType: DescType; modeFlags: LongInt;
VAR resultingText: AEDesc): OSAError;
CHAPTER 10

Scripting Components

Manipulating Dialects

FUNCTION OSASetCurrentDialect
(scriptingComponent: ComponentInstance;
dialectCode: Integer): OSAError;

FUNCTION OSAGetCurrentDialect
(scriptingComponent: ComponentInstance;
VAR resultingDialectCode: Integer): OSAError;

FUNCTION OSAAvailableDialectCodeList
(scriptingComponent: ComponentInstance;
VAR resultingDialectCodeList: AEDesc)
: OSAError;

FUNCTION OSAGetDialectInfo
(scriptingComponent: ComponentInstance;
dialectCode: Integer; selector: OSType;
VAR resultingDialectInfo: AEDesc): OSAError;

FUNCTION OSAAvailableDialects
(scriptingComponent: ComponentInstance;
VAR resultingDialectCodeList: AEDesc)
: OSAError;

Using Script Contexts to Handle Apple Events

FUNCTION OSASetResumeDispatchProc
(scriptingComponent: ComponentInstance;
resumeDispatchProc: AEHandlerProcPtr;
refCon: LongInt): OSAError;

FUNCTION OSAGetResumeDispatchProc
(scriptingComponent: ComponentInstance;
VAR resumeDispatchProc: AEHandlerProcPtr;
VAR refCon: LongInt): OSAError;

FUNCTION OSAExecuteEvent
(scriptingComponent: ComponentInstance;
theAppleEvent: AppleEvent;
contextID: OSAID; modeFlags: LongInt;
VAR resultingScriptValueID: OSAID): OSAError;

FUNCTION OSADoEvent
(scriptingComponent: ComponentInstance;
theAppleEvent: AppleEvent;
contextID: OSAID; modeFlags: LongInt;
VAR reply: AppleEvent): OSAError;

FUNCTION OSAMakeContext
(scriptingComponent: ComponentInstance;
contextName: AEDesc;
parentContext: OSAID;
VAR resultingContextID: OSAID): OSAError;
CHAPTER 10
Scripting Components

AppleScript Component Routines

Initializing AppleScript

FUNCTION ASInit (scriptingComponent: ComponentInstance; modeFlags: LongInt; minStackSize: LongInt; preferredStackSize: LongInt; maxStackSize: LongInt; minHeapSize: LongInt; preferredHeapSize: LongInt; maxHeapSize: LongInt): OSAError;

Getting and Setting Styles for Source Data

FUNCTION ASGetSourceStyles (scriptingComponent: ComponentInstance; VAR resultingSourceStyles: STHandle): OSAError;
FUNCTION ASSetSourceStyles (scriptingComponent: ComponentInstance; sourceStyles: STHandle): OSAError;
FUNCTION ASGetSourceStyleNames (scriptingComponent: ComponentInstance; modeFlags: LongInt; VAR resultingSourceStyleNamesList: AEDescList) : OSAError;

Generic Scripting Component Routines

Getting and Setting the Default Scripting Component

FUNCTION OSAGetDefaultScriptingComponent (genericScriptingComponent: ComponentInstance; VAR scriptingSubType: ScriptingComponentSelector): OSAError;
FUNCTION OSASetDefaultScriptingComponent (genericScriptingComponent: ComponentInstance; scriptingSubType: ScriptingComponentSelector): OSAError;

Using Component-Specific Routines

FUNCTION OSAGetScriptingComponentFromStored (genericScriptingComponent: ComponentInstance; scriptData: AEDesc; VAR scriptingSubType: ScriptingComponentSelector): OSAError;
CHAPTER 10

Scripting Components

FUNCTION OSAGetScriptingComponent
(genericScriptingComponent: ComponentInstance;
scriptingSubType: ScriptingComponentSelector;
VAR scriptingInstance: ComponentInstance)
: OSAError;

FUNCTION OSGenericToRealID (genericScriptingComponent: ComponentInstance;
VAR theScriptID: OSAID;
VAR theExactComponent: ComponentInstance)
: OSAError;

FUNCTION OSARealToGenericID (genericScriptingComponent: ComponentInstance;
VAR theScriptID: OSAID;
theExactComponent: ComponentInstance)
: OSAError;

Routines Used by Scripting Components

Manipulating Trailers for Generic Storage Descriptor Records
FUNCTION OSAGetStorageType (scriptData: Handle; VAR type: DescType): OSErr;
FUNCTION OSAAddStorageType (scriptData: Handle; type: DescType): OSErr;
FUNCTION OSARemoveStorageType (scriptData: Handle): OSErr;

Application-Defined Routines

FUNCTION MyActiveProc (refCon: LongInt): OSErr;
FUNCTION MyAECreateProc (theAEEventClass: AEEventClass;
theAEEventID: AEEventID; target: AEAddressDesc;
returnID: Integer; transactionID: LongInt;
VAR result: AppleEvent;
refCon: LongInt): OSErr;

FUNCTION MyAESendProc (theAppleEvent: AppleEvent;
VAR reply: AppleEvent; sendMode: AESendMode;
sendPriority: AESendPriority;
timeOutInTicks: LongInt;
idleProc: IdleProcPtr;
filterProc: EventFilterProcPtr;
refCon: LongInt): OSErr;

FUNCTION MyResumeDispatch (theAppleEvent: AppleEvent; reply: AppleEvent;
refCon: LongInt): OSErr;
C Summary

Constants

/*Component Manager type and subtype codes*/
#define kOSAComponentType 'osa '
#define kOSAGenericScriptingComponentSubtype 'scpt'

/*null script ID passed to OSAExecute, OSAGetSource, or OSAScriptRecording*/
#define kOSANullScript ((OSAID) 0)

/*mode flag that indicates a routine's default mode settings are to be used*/
#define kOSAModeNull 0
#define kOSANullMode 0

/*mode flag used with OSAScript to store a script context without storing */
/*its parent context*/
#define kOSAModeDontStoreParent 0x00004000

/*mode flag used with OSAScript, OSALoad, or OSACscript to request */
/*minimum script data*/
#define kOSAModePreventGetSource 0x00000001

/*mode flags used with OSAScript, OSAScriptContext, OSAScriptExecute, */
/*OSAScriptCompile, OSAScriptExecuteEvent, and OSAScriptEvent*/

/*these mode flags cause the scripting component to set the corresponding */
/*sendMode flags when it sends the event*/
#define kOSAModeNeverInteract kAENeverInteract
#define kOSAModeCanInteract kAECanInteract
#define kOSAModeAlwaysInteract kAEAlwaysInteract
#define kOSAModeDontReconnect kAEDontReconnect

/*this mode flag causes the scripting component not to set the */
/*kAECanSwitchLayer sendMode flag when it sends the event*/
#define kOSAModeDontSwitchLayer 0x00000040

/*this mode flag causes the scripting component not to set the */
/*kAE contributions sendMode flag when it sends the event*/
#define kOSAModeDoRecord 0x00001000
/* mode flags used with OSACompile*/

/* this mode flag causes OSACompile to compile the source data as a script */
/* context*/
#define kOSAModeCompileIntoContext 0x00000002

/* this mode flag causes OSACompile to augment the script data for a script */
/* context rather than replacing it*/
#define kOSAModeAugmentContext 0x00000004

/* mode flags used with OSADisplay or OSADoScript to indicate that output */
/* needs to be readable by humans only and does not have to be recompiled */
/* by OSACompile*/
#define kOSAModeDisplayForHumans 0x00000008

/* suite and event code for the Recorded Text event*/
#define kOSASuite 'ascr'
#define kOSARecordedText 'recd'

/* resource type for stored script data*/
#define kOSAScriptResourceType kOSAGenericScriptingComponentSubtype

/* descriptor type for generic storage descriptor records*/
#define typeOSAGenericStorage kOSAScriptResourceType

/* descriptor types and error range keywords for OSAScriptError*/
#define kOSAErrorNumber 'errn' /*returns error number*/
#define kOSAErrorMessage 'errs' /*returns error message*/
#define kOSAErrorBriefMessage 'errb' /*returns brief error*/
                         /* message*/
#define kOSAErrorApp 'erap' /*returns PSN or name of*/
                        /* errant application*/
#define kOSAErrorPartialResult 'ptlr' /*returns partial result, */
                                  /* if any*/
#define kOSAErrorOffendingObject 'erob' /*returns info about*/
                                   /* offending object, if any*/
#define kOSAErrorRange 'erng' /*returns error range*/
#define typeOSAErrorRange 'erng' /*descriptor type for*/
                        /* error range*/
#define keyOSASourceStart 'srcs' /*start of error range*/
#define keyOSASourceEnd 'srce' /*end of error range*/
/*if selector parameter of kOSAScriptError is kOSAErrorNumber, scripting */
/* components may return any of these error codes*/

/*dynamic errors*/
#define errOSACantCoerce errAECoercionFail
#define errOSACantAccess errAENoSuchObject
#define errOSAGeneralError -2700
#define errOSADivideByZero -2701
#define errOSANumericOverflow -2702
#define errOSACantLaunch -2703
#define errOSAAppNotHighLevelEventAware -2704
#define errOSACorruptTerminology -2705
#define errOSAStackOverflow -2706
#define errOSAInternalTableOverflow -2707
#define errOSADataBlockTooLarge -2708

/*component-specific dynamic script errors: -2720 through -2739*/

/*static errors*/
#define errTypeError errAEWrongDataType
#define errOSAMessageNotUnderstood errAEEventNotHandled
#define errOSAUndefinedMessage errAEHandlerNotFound
#define errOSAIllegalIndex errAEImpossibleRange
#define errOSASyntaxError -2740
#define errOSASyntaxTypeError -2741
#define errOSATokenTooLong -2742
#define errOSAMissingParameter errAEDescNotFound
#define errOSAParameterMismatch errAEWrongNumberArgs
#define errOSADuplicateParameter -2750
#define errOSADuplicateProperty -2751
#define errOSADuplicateHandler -2752
#define errOSAUndefinedVariable -2753
#define errOSAInconsistentDeclarations -2754
#define errOSAControlFlowError -2755

/*component-specific static script errors: -2760 through -2779*/

/*dialect-specific script errors: -2780 through -2799*/

/*descriptor type for each item in list returned by OSAAvailableDialects*/
#define typeOSADialectInfo 'difo'
/*keywords for descriptor record of descriptor type typeOSADialectInfo; */
/* these can also be used in selector parameter of OSAGetDialectInfo*/
#define keyOSADialectName 'dnam' /*used with descriptor */
/* record of any text */
/* type, such as typeChar*/
#define keyOSADialectCode 'dcod' /*used with descriptor */
/* record of type */
/* typeShortInteger*/
#define keyOSADialectLangCode 'dlcd' /*used with descriptor */
/* record of type */
/* typeShortInteger*/
#define keyOSADialectScriptCode 'dscd' /*used with descriptor */
/* record of type */
/* typeShortInteger*/

/*constants for use with OSASetResumeDispatchProc*/
#define kOSAUseStandardDispatch kAEUseStandardDispatch
#define kOSANoDispatch kAENoDispatch
#define kOSADontUsePhac $0001

/*selectors for use with OSAGetScriptInfo*/
#define kOSAScriptIsModified 'modi'
#define kOSAScriptIsTypeCompiledScript 'cscr'
#define kOSAScriptIsTypeScriptValue 'valu'
#define kOSAScriptIsTypeScriptContext 'cntx'
#define kOSAScriptBestType 'best'
#define kOSACanGetSource 'gsrc'

/*OSA component flags*/
#define kOSASupportsCompiling 0x0002
#define kOSASupportsGetSource 0x0004
#define kOSASupportsAECoercion 0x0008
#define kOSASupportsAESending 0x0010
#define kOSASupportsRecording 0x0020
#define kOSASupportsConvenience 0x0040
#define kOSASupportsDialects 0x0080
#define kOSASupportsEventHandling 0x0100

/*component selectors*/
/*basic scripting*/
#define kOSASelectLoad 0x0001
CHAPTER 10

Scripting Components

#define kOSASelectStore 0x0002
#define kOSASelectExecute 0x0003
#define kOSASelectDisplay 0x0004
#define kOSASelectScriptError 0x0005
#define kOSASelectDispose 0x0006
#define kOSASelectSetScriptInfo 0x0007
#define kOSASelectGetScriptInfo 0x0008
#define kOSASelectSetActiveProc 0x0009
#define kOSASelectGetActiveProc 0x000A

/*compiling*/
#define kOSASelectScriptingComponentName 0x0102
#define kOSASelectCompile 0x0103
#define kOSASelectCopyID 0x0104

/*getting source data*/
#define kOSASelectGetSource 0x0201

/*coercing script values*/
#define kOSASelectCoerceFromDesc 0x0301
#define kOSASelectCoerceToDesc 0x0302

/*manipulating send and create functions*/
#define kOSASelectSetSendProc 0x0401
#define kOSASelectGetSendProc 0x0402
#define kOSASelectSetCreateProc 0x0403
#define kOSASelectGetCreateProc 0x0404
#define kOSASelectSetDefaultTarget 0x0405

/*recording*/
#define kOSASelectStartRecording 0x0501
#define kOSASelectStopRecording 0x0502

/*convenience*/
#define kOSASelectLoadExecute 0x0601
#define kOSASelectCompileExecute 0x0602
#define kOSASelectDoScript 0x0603

/*manipulating dialects*/
#define kOSASelectSetCurrentDialect 0x0701
#define kOSASelectGetCurrentDialect 0x0702
#define kOSASelectAvailableDialects 0x0703
#define kOSASelectGetDialectInfo 0x0704

10-116  Summary of Scripting Components
CHAPTER 10

Scripting Components

#define kOSASelectAvailableDialectCodeList 0x0705

/*executing Apple event handlers in script contexts*/
#define kOSASelectSetResumeDispatchProc 0x0801
#define kOSASelectGetResumeDispatchProc 0x0802
#define kOSASelectExecuteEvent 0x0803
#define kOSASelectDoEvent 0x0804
#define kOSASelectMakeContext 0x0805

/*scripting-component-specific selectors are added beginning with this */
/* value*/
#define kOSASelectComponentSpecificStart 0x1001

/**********AppleScript component constants**********/
#define typeAppleScript 'ascr'

/*Component Manager subtype for AppleScript component*/
#define kAppleScriptSubtype typeAppleScript

/*AppleScript constant for storage descriptor records*/
#define typeASStorage typeAppleScript

/*AppleScript constant for the selector parameter of OSAGetScriptInfo*/
#define kASHasOpenHandler 'hsod'

/*AppleScript component selectors*/
#define kASSelectInit 0x1001
#define kASSelectSetSourceStyles 0x1002
#define kASSelectGetSourceStyles 0x1003
#define kASSelectGetSourceStyleNames 0x1004

/*default initialization parameters for AppleScript*/
#define kASDefaultMinStackSize 1 * 1024
#define kASDefaultPreferredStackSize 4 * 1024
#define kASDefaultMaxStackSize 16 * 1024
#define kASDefaultMinHeapSize 4 * 1024
#define kASDefaultPreferredHeapSize 64 * 1024
#define kASDefaultMaxHeapSize 32 * 1024 * 1024

/*AppleScript source style flags*/
#define kASSourceStyleUncompiledText 0
#define kASSourceStyleNormalText 1
#define kASSourceStyleLanguageKeyword 2
#define kASSourceStyleApplicationKeyword 3
#define kASSourceStyleComment 4
#define kASSourceStyleLiteral 5
#define kASSourceStyleUserSymbol 6
#define kASSourceStyleObjectSpecifier 7
#define kASNumberOfSourceStyles 8

/*if selector parameter of kOSAScriptError is kOSAErrorNumber, AppleScript */
/* component may return any of these error codes*/
#define errASCantConsiderAndIgnore -2720
#define errASCantCompareMoreThan32k -2721
#define errASCantCompareMixedScripts -2722
#define errASTerminologyNestingTooDeep -2760
#define errASInconsistentNames -2780 /*English dialect*/

/******************generic scripting component constants**********/

/*component version this header file describes*/
kGenericComponentVersion 0x0100

/*generic scripting component selectors*/
#define kGSSSelectGetDefaultScriptingComponent 0x1001
#define kGSSSelectSetDefaultScriptingComponent 0x1002
#define kGSSSelectGetScriptingComponent 0x1003
#define kGSSSelectGetScriptingSystemFromStored 0x1004
#define kGSSSelectGenericToRealID 0x1005
#define kGSSSelectRealToGenericID 0x1006

Data Types

typedef unsigned long OSAID; /*script ID*/
typedef ComponentResult OSAError; /*type for result codes*/

/*pointers for application-defined functions*/
typedef pascal OSErr (*OSAActiveProcPtr) (long refCon);
typedef pascal OSErr (*AESendProcPtr)
    (const AppleEvent* theAppleEvent,
    AppleEvent* reply, AESendMode sendMode,
    AESendPriority sendPriority,
    long timeOutInTicks, IdleProcPtr idleProc,
    EventFilterProcPtr filterProc, long refCon);
CHAPTER 10
Scripting Components

typedef pascal OSErr (*AECreateAppleEventProcPtr) (
    AEEventClass theAEEventClass,
    AEEventID theAEEventID,
    const AEAddressDesc* target, short returnID,
    long transactionID, AppleEvent* result,
    long refCon);

typedef pascal OSErr (*AEHandlerProcPtr) (
    const AppleEvent* theAppleEvent,
    AppleEvent* reply, long refCon);

/*generic scripting component data types*/
typedef OSType ScriptingComponentSelector;
typedef OSAID GenericID;

Required Scripting Component Routines

Saving and Loading Script Data
pascal OSAError OSAStore (ComponentInstance scriptingComponent, 
    OSAID scriptID, DescType desiredType, 
    long modeFlags, AEDesc* resultingScriptData);
pascal OSAError OSALoad (ComponentInstance scriptingComponent, 
    const AEDesc* scriptData, long modeFlags, 
    OSAID* resultingScriptID);

Executing and Disposing of Scripts
pascal OSAError OSAExecute (ComponentInstance scriptingComponent, 
    OSAID compiledScriptID, OSAID contextID, 
    long modeFlags, OSAID* resultingScriptValueID);
pascal OSAError OSADisplay (ComponentInstance scriptingComponent, 
    OSAID scriptValueID, DescType desiredType, 
    long modeFlags, AEDesc* resultingText);
pascal OSAError OSAScriptError (ComponentInstance scriptingComponent, 
    OSType selector, DescType desiredType, 
    AEDesc* resultingErrorDescription);
pascal OSAError OSADispose (ComponentInstance scriptingComponent, 
    OSAID scriptID);

Setting and Getting Script Information
pascal OSAError OSASetScriptInfo (ComponentInstance scriptingComponent, 
    OSAID scriptID, OSType selector, long value);
CHAPTER 10

Scripting Components

pascal OSAError OSAGetScriptInfo
    (ComponentInstance scriptingComponent,
     OSAID scriptID, OSType selector, long* result);

Manipulating the Active Function
pascal OSAError OSASetActiveProc
    (ComponentInstance scriptingComponent,
     OSAActiveProcPtr activeProc, long refCon);
pascal OSAError OSAGetActiveProc
    (ComponentInstance scriptingComponent,
     OSAActiveProcPtr* activeProc, long* refCon);

Optional Scripting Component Routines

Compiling Scripts
pascal OSAError OSAScriptingComponentName
    (ComponentInstance scriptingComponent,
     AEDesc* resultingScriptingComponentName);
pascal OSAError OSACompile
    (ComponentInstance scriptingComponent,
     const AEDesc* sourceData,
     long modeFlags,
     OSAID* previousAndResultingScriptID);
pascal OSAError OSACopyID
    (ComponentInstance scriptingComponent,
     OSAID fromID, OSAID* toID);

Getting Source Data
pascal OSAError OSAGetSource
    (ComponentInstance scriptingComponent,
     OSAID scriptID, DescType desiredType,
     AEDesc* resultingSourceData);

Coercing Script Values
pascal OSAError OSACoerceFromDesc
    (ComponentInstance scriptingComponent,
     const AEDesc* scriptData, long modeFlags,
     OSAID* resultingScriptValueID);
pascal OSAError OSACoerceToDesc
    (ComponentInstance scriptingComponent,
     OSAID scriptValueID, DescType desiredType,
     long modeFlags, AEDesc* result);
CHAPTER 10

Scripting Components

Manipulating the Create and Send Functions

pascal OSAError OSASetCreateProc
(ComponentInstance scriptingComponent,
AECreateAppleEventProcPtr createProc,
long refCon);

pascal OSAError OSAGetCreateProc
(ComponentInstance scriptingComponent,
AECreateAppleEventProcPtr* createProc,
long* refCon);

pascal OSAError OSASetSendProc
(ComponentInstance scriptingComponent,
AESendProcPtr sendProc, long refCon);

pascal OSAError OSAGetSendProc
(ComponentInstance scriptingComponent,
AESendProcPtr* sendProc, long* refCon);

pascal OSAError OSASetDefaultTarget
(ComponentInstance scriptingComponent,
const AEAddressDesc* target);

Recording Scripts

pascal OSAError OSAStrartRecording
(ComponentInstance scriptingComponent,
OSAID* compiledScriptToModifyID);

pascal OSAError OSAStropRecording
(ComponentInstance scriptingComponent,
OSAID compiledScriptID);

Executing Scripts in One Step

pascal OSAError OSALoadExecute
(ComponentInstance scriptingComponent,
const AEDesc* scriptData, OSAID contextID,
long modeFlags, OSAID* resultingScriptValueID);

pascal OSAError OSACompileExecute
(ComponentInstance scriptingComponent,
const AEDesc* sourceData, OSAID contextID,
long modeFlags, OSAID* resultingScriptValueID);

pascal OSAError OSADoScript
(ComponentInstance scriptingComponent,
const AEDesc* sourceData, OSAID contextID,
DescType desiredType, long modeFlags,
AEDesc* resultingText);
Manipulating Dialects

```pascal
OSAError OSASetCurrentDialect
(ComponentInstance scriptingComponent,
 short dialectCode);

OSAError OSAGetCurrentDialect
(ComponentInstance scriptingComponent,
 short* resultingDialectCode);

OSAError OSAAvailableDialectCodeList
(ComponentInstance scriptingComponent,
 AEDesc* resultingDialectCodeList);

OSAError OSAGetDialectInfo
(ComponentInstance scriptingComponent,
 short dialectCode, OSTYPE selector,
 AEDesc* resultingDialectInfo);

OSAError OSAAvailableDialects
(ComponentInstance scriptingComponent,
 AEDesc* resultingDialectInfoList);
```

Using Script Contexts to Handle Apple Events

```pascal
OSAError OSASetResumeDispatchProc
(ComponentInstance scriptingComponent,
 AEHandlerProcPtr resumeDispatchProc,
 long refCon);

OSAError OSAGetResumeDispatchProc
(ComponentInstance scriptingComponent,
 AEHandlerProcPtr* resumeDispatchProc,
 long* refCon);

OSAError OSExecuteEvent
(ComponentInstance scriptingComponent,
 const AppleEvent* theAppleEvent,
 OSAID contextID, long modeFlags,
 OSAID* resultingScriptValueID);

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

OSAMakeContext (ComponentInstance scriptingComponent,
 const AEDesc* contextName,
 OSAID parentContext,
 OSAID* resultingContextID);
```
AppleScript Component Routines

Initializing AppleScript

```pascal
OSAError ASInit
(ComponentInstance scriptingComponent,
long modeFlags, long minStackSize,
long preferredStackSize, long maxStackSize,
long minHeapSize, long preferredHeapSize,
long maxHeapSize);
```

Getting and Setting Styles for Source Data

```pascal
OSAError ASGetSourceStyles
(ComponentInstance scriptingComponent,
STHandle* resultingSourceStyles);

OSAError ASSetSourceStyles
(ComponentInstance scriptingComponent,
STHandle sourceStyles);

OSAError ASGetSourceStyleNames
(ComponentInstance scriptingComponent,
long modeFlags,
AEDescList* resultingSourceStyleNamesList);
```

Generic Scripting Component Routines

Getting and Setting the Default Scripting Component

```pascal
OSAError OSAGetDefaultScriptingComponent
(ComponentInstance genericScriptingComponent,
ScriptingComponentSelector* scriptingSubType);

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

Using Component-Specific Routines

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

OSAError OSAGetScriptingComponent
(ComponentInstance genericScriptingComponent,
ScriptingComponentSelector scriptingSubType,
ComponentInstance* scriptingInstance);
```
CHAPTER 10

Scripting Components

pascal OSAError OSGenericToRealID
    (ComponentInstance genericScriptingComponent,
     OSAID *theScriptID,
     ComponentInstance *theExactComponent);

pascal OSAError OSARealToGenericID
    (ComponentInstance genericScriptingComponent,
     OSAID *theScriptID,
     ComponentInstance theExactComponent);

Routines Used by Scripting Components

Manipulating Trailers for Generic Storage Descriptor Records

pascal OSErr OSAGetStorageType
    (Handle scriptData, DescType* type);

pascal OSErr OSAAddStorageType
    (Handle scriptData, DescType type);

pascal OSErr OSARemoveStorageType
    (Handle scriptData);

Application-Defined Routines

pascal OSErr MyActiveProc (long refCon);

pascal OSErr MyAECreateProc
    (AEEventClass theAEEventClass,
     AEEventID theAEEventID, AEAddressDesc target,
     short returnID, long transactionID,
     AppleEvent* result, long refCon);

pascal OSErr MyAESendProc
    (AppleEvent theAppleEvent, AppleEvent* reply,
     AESendMode sendMode,
     AESendPriority sendPriority,
     long timeOutInTicks, IdleProcPtr idleProc,
     EventFilterProcPtr filterProc, long refCon);

pascal OSErr MyResumeDispatch
    (const AppleEvent* theAppleEvent,
     AppleEvent* reply, long refCon);
### Result Codes

<table>
<thead>
<tr>
<th>Code</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>noErr</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>errOSACantCoerce</td>
<td>-1700</td>
<td>Same as errAECoercionFail; data could not be coerced to the requested data type</td>
</tr>
<tr>
<td>errOSACorruptData</td>
<td>-1702</td>
<td>Same as errAECorruptData</td>
</tr>
<tr>
<td>errAEEventNotHandled</td>
<td>-1708</td>
<td>Event wasn’t handled by an Apple event handler</td>
</tr>
<tr>
<td>errAERecordingIsAlreadyOn</td>
<td>-1732</td>
<td>Attempt to turn recording on when it is already on for a recording process</td>
</tr>
<tr>
<td>errOSASystemError</td>
<td>-1750</td>
<td>General scripting system error</td>
</tr>
<tr>
<td>errOSAInvalidID</td>
<td>-1751</td>
<td>Invalid script ID</td>
</tr>
<tr>
<td>errOSABadStorageType</td>
<td>-1752</td>
<td>Illegal storage type</td>
</tr>
<tr>
<td>errOSAScriptError</td>
<td>-1753</td>
<td>Error occurred during compilation or execution</td>
</tr>
<tr>
<td>errOSABadSelector</td>
<td>-1754</td>
<td>Selector not supported by scripting component</td>
</tr>
<tr>
<td>errOSASourceNotAvailable</td>
<td>-1756</td>
<td>Source data not available</td>
</tr>
<tr>
<td>errOSANoSuchDialect</td>
<td>-1757</td>
<td>Invalid dialect code</td>
</tr>
<tr>
<td>errOSADATAFormatObsolete</td>
<td>-1758</td>
<td>Data format is obsolete</td>
</tr>
<tr>
<td>errOSADATAFormatTooNew</td>
<td>-1759</td>
<td>Data format is too new</td>
</tr>
<tr>
<td>errOSAComponentMismatch</td>
<td>-1761</td>
<td>Generic scripting component error; parameters are for two different scripting components instead of the same one</td>
</tr>
<tr>
<td>errOSACantOpenComponent</td>
<td>-1762</td>
<td>Generic scripting component error; can’t connect to scripting component</td>
</tr>
<tr>
<td>badComponentInstance</td>
<td>$80008001</td>
<td>Invalid component instance</td>
</tr>
</tbody>
</table>