Scripting Bridge Framework Reference

Interapplication Communication

2007-05-29

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

Simultaneously published in the United States and Canada.

Contents

Introduction

Framework	/System/Library/Frameworks/ScriptingBridge.framework
Header file directories	/System/Library/Frameworks/ScriptingBridge.framework/Headers
Declared in	SBApplication.h SBElementArray.h SBObject.h

Scripting Bridge is a technology that lets you control scriptable Apple and third-party applications using standard Objective-C syntax. Introduced in Mac OS X version 10.5 (Leopard), the Scripting Bridge framework dynamically implements an Objective-C bridge to OSA-compliant applications—that is, applications having a scripting interface (usually defined in a `sdef` file). As part of this implementation, it generates Objective-C class implementations of the classes it finds in the scripting interface, including objects and methods representing properties, elements, commands, and so on. The objects are derived from classes defined in the Scripting Bridge framework.

Introduction

Classes

PART I

Classes

SBApplication Class Reference

Inherits from	SBObject : NSObject
Conforms to	NSCoding NSCoding (SBObject) NSObject (NSObject)
Framework	/System/Library/Frameworks/ScriptingBridge.framework
Declared in	ScriptingBridge/SBApplication.h
Availability	Available in Mac OS X v10.5 and later
Related sample code	iChatStatusFromApplication SBSetFinderComment SBSystemPrefs ScriptingBridgeFinder ScriptingBridgeiCal

Overview

The `SBApplication` class provides a mechanism enabling an Objective-C program to send Apple events to a scriptable application and receive Apple events in response. It thereby makes it possible for that program to control the application and exchange data with it. Scripting Bridge works by bridging data types between Apple event descriptors and Cocoa objects.

Although `SBApplication` includes methods that manually send and process Apple events, you should never have to call these methods directly. Instead, subclasses of `SBApplication` implement application-specific methods that handle the sending of Apple events automatically.

For example, if you wanted to get the current iTunes track, you can simply use the `currentTrack` method of the dynamically defined subclass for the iTunes application—which handles the details of sending the Apple event for you—rather than figuring out the more complicated, low-level alternative:

```
[iTunes propertyWithCode:'pTrk'];
```

If you do need to send Apple events manually, consider using the `NSAppleEventDescriptor` class.

Subclassing Notes

You rarely instantiate `SBApplication` objects directly. Instead, you get the shared instance of a application-specific subclass typically by calling one of the `applicationWith...` class methods, using a bundle identifier, process identifier, or URL to identify the application.

Tasks

Getting a Scriptable Application Instance

`+ applicationWithBundleIdentifier:` (page 11)
Returns the shared instance representing the target application specified by its bundle identifier.

`+ applicationWithProcessIdentifier:` (page 12)
Returns the shared instance representing a target application specified by its process identifier.

`+ applicationWithURL:` (page 12)
Returns the shared instance representing a target application specified by the given URL.

Initializing a Scriptable Application Object

`- initWithBundleIdentifier:` (page 15)
Returns an instance of an `SBApplication` subclass that represents the target application identified by the given bundle identifier.

`- initWithProcessIdentifier:` (page 16)
Returns an instance of an `SBApplication` subclass that represents the target application identified by the given process identifier.

`- initWithURL:` (page 16)
Returns an instance of an `SBApplication` subclass that represents the target application identified by the given URL.

Creating a Scripting Class

`- classForScriptingClass:` (page 13)
Returns a class object that represents a particular class in the target application.

Controlling the Application

`- activate` (page 13)
Moves the target application to the foreground immediately.

`- isRunning` (page 17)
Returns whether the target application represented by the receiver is running.

`- launchFlags` (page 17)
Returns the launch flags for the application represented by the receiver.

- `setLaunchFlags:` (page 18)
  Returns the launch flags for the application represented by the receiver.
- `sendMode` (page 18)
  Returns the mode for sending Apple events to the target application.
- `setSendMode:` (page 19)
  Sets the mode for sending Apple events to the target application.
- `timeout` (page 20)
  Returns the period the application will wait to receive reply Apple events.
- `setTimeout:` (page 19)
  Sets the maximum time the application will wait to receive reply Apple events.

Getting Class Names and Codes

- `classNamesForCodes` (page 14)
  Returns a dictionary mapping four-character class codes to the names of their corresponding Objective-C classes.
- `codesForPropertyNames` (page 14)
  Returns a dictionary mapping property keys to their corresponding four-character codes.

Managing the Delegate

- `delegate` (page 15)
  Returns the error-handling delegate of the receiver.
- `setDelegate:` (page 18)
  Returns the error-handling delegate of the receiver.

Class Methods

applicationWithBundleIdentifier:

Returns the shared instance representing the target application specified by its bundle identifier.

```
+ (id)applicationWithBundleIdentifier:(NSString *)ident
```

Parameters

ident
  A bundle identifier specifying an application that is OSA-compliant.

Return Value

An instance of a `SBApplication` subclass that represents the target application whose bundle identifier is *ident*. Returns `nil` if no such application can be found or if the application does not have a scripting interface.

Discussion

For applications that declare themselves to have a dynamic scripting interface, this method will launch the application if it is not already running.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- initWithBundleIdentifier:` (page 15)

Related Sample Code
SBSendEmail
SBSetFinderComment
SBSystemPrefs
ScriptingBridgeFinder
ScriptingBridgeiCal

Declared In
`SBApplication.h`

applicationWithProcessIdentifier:

Returns the shared instance representing a target application specified by its process identifier.

```
+ (id)applicationWithProcessIdentifier:(pid_t)pid
```

Parameters

pid

The BSD process ID of a OSA-compliant application. Often you can get the process ID of a process using the `processIdentifier` method of `NSTask`.

Return Value
An instance of an `SBApplication` subclass that represents the target application whose process identifier is *pid*. Returns `nil` if no such application can be found or if the application does not have a scripting interface.

Discussion
You should avoid using this method unless you know nothing about a target application but its process ID. In most cases, it is better to use `classForApplicationWithBundleIdentifier:` (page 11), which will dynamically locate the application's path at runtime, or `classForApplicationWithURL:` (page 12), which is not dependent on the target application being open at the time the method is called.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- initWithProcessIdentifier:` (page 16)

Declared In
`SBApplication.h`

applicationWithURL:

Returns the shared instance representing a target application specified by the given URL.

```
+ (id)applicationWithURL:(NSURL *)url
```

Parameters

url

The Universal Resource Locator (URL) locating an OSA-compliant application.

Return Value

An `SBApplication` subclass from which to generate a shared instance of the target application whose URL is *url*. Returns `nil` if no such application can be found or if the application does not have a scripting interface.

Discussion

For applications that declare themselves to have a dynamic scripting interface, this method will launch the application if it is not already running. This approach to initializing `SBApplication` objects should be used only if you know for certain the URL of the target application. In most cases, it is better to use `classForApplicationWithBundleIdentifier:` (page 11) which dynamically locates the target application at runtime.

This method currently supports file URLs (`file:`) and remote application URLs (`eppc:`). It checks whether a file exists at the specified path, but it does not check whether an application identified via `eppc:` exists.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- initWithURL:` (page 16)

Declared In

`SBApplication.h`

Instance Methods

activate

Moves the target application to the foreground immediately.

```
- (void)activate
```

Discussion

If the target application is not already running, this method launches it.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SBApplication.h`

classForScriptingClass:

Returns a class object that represents a particular class in the target application.

```
- (Class)classForScriptingClass:(NSString *)className
```

Parameters

className

The name of the scripting class.

Return Value

A `Class` object representing the scripting class.

Discussion

You invoke this method on an instance of a scriptable application. Once you have the class object, you may allocate an instance of the class and appropriately the raw instance. Or you may use it in a call to `isKindOfClass:` to determine the class type of an object.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SBApplication.h`

classNamesForCodes

Returns a dictionary mapping four-character class codes to the names of their corresponding Objective-C classes.

```
- (NSDictionary *)classNamesForCodes
```

Return Value

A dictionary whose keys are four-character class codes of the external application (as `NSNumber` objects), and whose values are the names of the corresponding `SBObject` subclasses.

Discussion

The default implementation returns an empty dictionary. Application-specific subclasses return dictionaries tailored to the types of objects they support.

You should never call this method directly.

Availability

Available in Mac OS X v10.5 through Mac OS X v10.5.

Declared In

`SBApplication.h`

codesForPropertyNames

Returns a dictionary mapping property keys to their corresponding four-character codes.

```
- (NSDictionary *)codesForPropertyNames
```

Return Value

A dictionary whose keys are the keys of properties of the external application, and whose values are the corresponding four-character codes (as `NSNumber` objects).

Discussion

The default implementation returns an empty dictionary. Application-specific subclasses return dictionaries tailored to the types of objects they support.

You should never call this method directly.

Availability
Available in Mac OS X v10.5 through Mac OS X v10.5.

Declared In
SBApplication.h

delegate

Returns the error-handling delegate of the receiver.

```
- (id)delegate
```

Return Value
The object acting as error-handling delegate of the receiver.

Availability
Available in Mac OS X v10.5 and later.

See Also
- setDelegate: (page 18)

Declared In
SBApplication.h

initWithBundleIdentifier:

Returns an instance of an SBApplication subclass that represents the target application identified by the given bundle identifier.

```
- (id)initWithBundleIdentifier:(NSString *)ident
```

Parameters
ident
A bundle identifier specifying an application that is OSA-compliant.

Return Value
An initialized shared instance of an SBApplication subclass that represents a target application with the bundle identifier of *ident*. Returns nil if no such application can be found or if the application does not have a scripting interface.

Discussion
If you must initialize an SBApplication object explictly, you should use this initializer if possible; unlike initWithProcessIdentifier: (page 16) and initWithURL: (page 16), this method is not dependent on changeable factors such as the target application's path or process ID. Even so, you should rarely have to initialize an SBApplication object yourself; instead, you should initialize an application-specific subclass such as iTunesApplication.

Note that this method does not check whether an application with the given bundle identifier actually exists.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ applicationWithBundleIdentifier:` (page 11)

Declared In
`SBApplication.h`

initWithProcessIdentifier:

Returns an instance of an `SBApplication` subclass that represents the target application identified by the given process identifier.

```
- (id)initWithProcessIdentifier:(pid_t)pid
```

Parameters

pid

A BSD process ID specifying an application that is OSA-compliant. Often you can get the process ID of a process using the `processIdentifier` method of `NSTask`.

Return Value
An initialized `SBApplication` that you can use to communicate with the target application specified by the process ID. Returns `nil` if no such application can be found or if the application does not have a scripting interface.

Discussion
You should avoid using this method unless you know nothing about an external application but its PID. In most cases, it is better to use `initWithBundleIdentifier:` (page 15), which will dynamically locate the external application's path at runtime, or `initWithURL:` (page 16), which is not dependent on the external application being open at the time the method is called.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ applicationWithProcessIdentifier:` (page 12)

Declared In
`SBApplication.h`

initWithURL:

Returns an instance of an `SBApplication` subclass that represents the target application identified by the given URL.

```
- (id)initWithURL:(NSURL *)url
```

Parameters

url

A Universal Resource Locator (URL) specifying an application that is OSA-compliant.

Return Value
An initialized `SBApplication` that you can use to communicate with the target application specified by the process ID. Returns `nil` if an application could not be found or if the application does not have a scripting interface.

Discussion
This approach to initializing `SBApplication` objects should be used only if you know for certain the URL of the target application. In most cases, it is better to use `classForApplicationWithBundleIdentifier:` (page 11) which dynamically locates the target application at runtime. Even so, you should rarely have to initialize an `SBApplication` yourself.

This method currently supports file URLs (`file:`) and remote application URLs (`eppc:`). It checks whether a file exists at the specified path, but it does not check whether an application identified via `eppc:` exists.

Availability
Available in Mac OS X v10.5 and later.

See Also
`+ applicationWithURL:` (page 12)

Declared In
`SBApplication.h`

isRunning

Returns whether the target application represented by the receiver is running.

```
- (BOOL)isRunning
```

Return Value
`YES` if the application is running, `NO` otherwise.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SBApplication.h`

launchFlags

Returns the launch flags for the application represented by the receiver.

```
- (LSLaunchFlags)launchFlags
```

Return Value
A mask specifying the launch flags that are used when the target application is launched. For more information, see *Launch Services Reference*.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- setLaunchFlags:` (page 18)

Declared In
`SBApplication.h`

sendMode

Returns the mode for sending Apple events to the target application.

```
- (AESendMode)sendMode
```

Return Value
A mask specifying the mode for sending Apple events to the target application. For more information, see *Apple Event Manager Reference*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setSendMode:` (page 19)

Declared In
`SBApplication.h`

setDelegate:

Returns the error-handling delegate of the receiver.

```
- (void)setDelegate:(id)delegate
```

Parameters
delegate
The object acting as delegate of the receiver.

Discussion
The delegate should implement the `eventDidFail:withError:` (page 37) method of the `SBApplicationDelegate` informal protocol.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `delegate` (page 15)

Declared In
`SBApplication.h`

setLaunchFlags:

Returns the launch flags for the application represented by the receiver.

```
- (void)setLaunchFlags:(LSLaunchFlags)flags
```

Parameters
flags
A mask specifying the launch flags that are used when the target application is launched. For more information, see *Launch Services Reference*.

Discussion
The default `SBApplication` launch flags are `kLSLaunchDontAddToRecents` (so the target application is not added to the Recent Items menu), `kLSLaunchDontSwitch` (so the target application launches in the background), and `kLSLaunchAndHide` (so the target application is hidden as soon as it is launched).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `launchFlags` (page 17)

Declared In
`SBApplication.h`

setSendMode:

Sets the mode for sending Apple events to the target application.

```
- (void)setSendMode:(AESendMode)sendMode
```

Parameters

`sendMode`

A mask specifying the mode for sending Apple events to the target application. For a list of valid modes, see *Apple Event Manager Reference*.

Discussion
The default send mode is `kAEWaitReply`. If the send mode is something other than `kAEWaitReply`, the receiver might not correctly handle reply events from the target application.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `sendMode` (page 18)

Declared In
`SBApplication.h`

setTimeout:

Sets the maximum time the application will wait to receive reply Apple events.

```
- (void)setTimeout:(long)timeout
```

Parameters

`timeout`

The time in ticks that the receiver will wait to receive a reply Apple event from the target application before giving up.

Discussion
The default timeout value is `kAEDefaultTimeout`, which is about a minute. If you want the receiver to wait indefinitely for reply Apple events, use `kNoTimeOut`. For more information, see *Apple Event Manager Reference*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `timeout` (page 20)

Declared In
`SBApplication.h`

timeout

Returns the period the application will wait to receive reply Apple events.

```
- (long)timeout
```

Return Value
The time in ticks that the receiver will wait to receive a reply Apple event from the target application before giving up. For more information, see *Apple Event Manager Reference*.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `setTimeout:` (page 19)

Declared In
`SBApplication.h`

SBElementArray Class Reference

Inherits from	NSMutableArray : NSArray : NSObject
Conforms to	NSCoding (NSArray) NSCopying (NSArray) NSMutableCopying (NSArray) NSFastEnumeration (NSArray) NSObject (NSObject)
Framework	/System/Library/Frameworks/ScriptingBridge.framework
Availability	Available in Mac OS X v10.5 and later.
Declared in	ScriptingBridge/SBElementArray.h
Related sample code	iChatStatusFromApplication SBSetFinderComment ScriptingBridgeFinder ScriptingBridgeiCal

Overview

`SBElementArray` is subclass of `NSMutableArray` that manages collections of related `SBObject` objects. For example, when you ask the Finder for a list of disks, or ask iTunes for a list of playlists, you get the result back as an `SBElementArray` containing Scripting Bridge objects representing those items.

`SBElementArray` defines methods beyond those of `NSArray` for obtaining individual objects. In addition to `objectAtIndex:`, `SBElementArray` also defines `objectWithName:` (page 25), `objectWithID:` (page 24), and `objectAtLocation:` (page 24).

Subclassing Notes

The `SBElementArray` class is not designed for subclassing.

Tasks

Getting Objects in the Array

- `objectWithName:` (page 25)
  Returns the object in the array with the given name.
- `objectWithID:` (page 24)
  Returns the object in the array with the given identifier.
- `objectAtLocation:` (page 24)
  Returns the object at the given location in the receiver.

Getting the Referenced Array

- `get` (page 23)
  Forces evaluation of the receiver, causing the real object to be returned immediately.

Filtering an Element Array

- `arrayByApplyingSelector:` (page 22)
  Returns a array containing the results of sending the specified message to each object in the receiver.
- `arrayByApplyingSelector:withObject:` (page 23)
  Returns a array containing the results of sending the specified message to each object in the receiver.

Instance Methods

arrayByApplyingSelector:

Returns a array containing the results of sending the specified message to each object in the receiver.

```
- (NSArray *)arrayByApplyingSelector:(SEL)selector
```

Parameters

selector
  A selector identifying the message to be sent to each object in the array.

Return Value
A new array containing the results of sending the *selector* message to each object in the receiver, starting with the first object and continuing through the element array to the last object.

Discussion
The method identified by *selector* must not take any arguments and must return an Objective-C object. It should not have the side effect of modifying the receiving array. The order of the items in the result array corresponds to the order of the items in the original array.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `arrayByApplyingSelector:withObject:` (page 23)

Declared In
`SBElementArray.h`

arrayByApplyingSelector:withObject:

Returns a array containing the results of sending the specified message to each object in the receiver.

```
- (NSArray *)arrayByApplyingSelector:(SEL)selector withObject:(id)argument
```

Parameters

selector
A selector identifying the message to be sent to each object in the array.

argument
The value for the parameter of the message identified by *selector*.

Return Value
A new array containing the results of sending the *selector* message to each object in the receiver, starting with the first object and continuing through the element array to the last object.

Discussion
The method identified by *selector* must take a single argument—whose value is provided in *argument*—and must return an Objective-C object. It should not have the side effect of modifying the receiving array. The order of the items in the result array corresponds to the order of the items in the original array.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `arrayByApplyingSelector:` (page 22)

Declared In
`SBElementArray.h`

get

Forces evaluation of the receiver, causing the real object to be returned immediately.

```
- (NSArray *)get
```

Return Value
The object referenced by the receiver.

Discussion
This method forces the evaluation of the current object reference (the receiver), resulting in the return of the referenced object. By default, Scripting Bridge deals with references to objects until you actually request some concrete data from them or until you call the `get` method.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SBElementArray.h`

objectAtLocation:

Returns the object at the given location in the receiver.

```
- (id) objectAtLocation:(id)loc
```

Parameters

loc

An object that specifies the absolute position of the object within the array. It can be an integer index, a list of coordinates, a URL, or other determinant. See the discussion for clarification.

Return Value
A reference to the `SBObject` object identified by *loc* or `nil` if the object couldn't be located.

Discussion
This method is a generalization of `objectAtIndex:` for applications where the "index" is not simply an integer. For example, Finder can specify objects using a `NSURL` object as a location. In OSA this is known as "absolute position," a generalization of the notion of "index" in Foundation—it could be an integer, but it doesn't have to be. A single object may even have a number of different "absolute position" values depending on the container.

Availability
Available in Mac OS X v10.5 and later.

See Also
`- objectWithName:` (page 25)
`- objectWithID:` (page 24)

Related Sample Code
SBSetFinderComment

Declared In
`SBElementArray.h`

objectWithID:

Returns the object in the array with the given identifier.

```
- (id)objectWithID:(id)identifier
```

Parameters

identifier

The identifier of one of the receiver's objects.

Return Value
A reference to the identified object or `nil` if could not be found.

Discussion
This method is provided as an alternative to `objectAtIndex:` for applications where an identifier is available instead of (or in addition to) an index. A unique ID is generally more stable than an index. For example, it may be more useful to identify a contact in Address Book by its identifier (which doesn't change over time) than by its index in the list of contacts (which can change as contacts are added or removed).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `objectWithName:` (page 25)
- `objectAtLocation:` (page 24)

Declared In
`SBElementArray.h`

objectWithName:

Returns the object in the array with the given name.

```
- (id)objectWithName:(NSString *)name
```

Parameters

name

The name of one of the receiver's objects.

Return Value
A reference to the designated object or `nil` if the object couldn't be found.

Discussion
This method is provided as an alternative to`objectAtIndex:` for applications where a name is available instead of (or in addition to) an index. A name is generally more stable than an index. For example, it is typically more useful to identify a mailbox in Mail by its name than by its index in the list of mailboxes.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `objectWithID:` (page 24)
- `objectAtLocation:` (page 24)

Declared In
`SBElementArray.h`

SBObject Class Reference

Inherits from	NSObject
Conforms to	NSCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/ScriptingBridge.framework
Declared in	ScriptingBridge/SBObject.h
Availability	Available in Mac OS X v10.5
Related sample code	iChatStatusFromApplication SBSetFinderComment ScriptingBridgeFinder ScriptingBridgeiCal

Overview

The `SBObject` class declares methods that can be invoked on any object in a scriptable application. It defines methods for getting elements and properties of an object, as well as setting a given object to a new value.

Each `SBObject` is built around an object specifier, which tells Scripting Bridge how to locate the object. Therefore, you can think of an `SBObject` as a reference to an object in an target application rather than an object itself. To bypass this reference-based approach and force evaluation, use the `get` (page 29) method.

Typically, rather than create `SBObject` instances explictly, you receive `SBObject` objects by calling methods of an `SBApplication` subclass. For example, if you wanted to get an `SBObject` representing the current iTunes track, you would use code like this (where `iTunesTrack` is a subclass of `SBObject`):

```
iTunesApplication *iTunes = [SBApplication
applicationWithBundleIdentifier:@"com.apple.iTunes"];
iTunesTrack *track = [iTunes currentTrack];
```

You can discover the names of dynamically generated classes such as `iTunesApplication` and `iTunesTrack` by examining the header file created by the `sdp` tool. Alternatively, you give these variables the dynamic Objective-C type `id`.

Tasks

Initializing a Scripting Bridge Object

- `init` (page 29)
  Initializes and returns an instance of an `SBObject` subclass.
- `initWithData:` (page 30)
  Returns an instance of an `SBObject` subclass initialized with the given data.
- `initWithProperties:` (page 31)
  Returns an instance of an `SBObject` subclass initialized with the specified properties.
- `initWithElementCode:properties:data:` (page 30)
  Returns an instance of an `SBObject` subclass initialized with the specified properties and data and added to the designated element array.

Getting Referenced Data

- `get` (page 29)
  Forces evaluation of the receiver, causing the real object to be returned immediately.

Sending Apple Events

- `sendEvent:id:parameters:` (page 33)
  Sends an Apple event with the given event class, event ID, and format to the target application.
- `setTo:` (page 33)
  Sets the receiver to a specified value.

Getting Properties and Elements

- `propertyWithClass:code:` (page 32)
  Returns an object of the designated scripting class representing the specified property of the receiver
- `propertyWithCode:` (page 32)
  Returns an object representing the specified property of the receiver.
- `elementArrayWithCode:` (page 28)
  Returns an array containing every child of the receiver with the given class-type code.

Instance Methods

elementArrayWithCode:

Returns an array containing every child of the receiver with the given class-type code.

```
- (SBElementArray *)elementArrayWithCode:(DescType)code
```

Parameters

code

A four-character code that identifies a scripting class.

Return Value

An `SBElementArray` object containing every child of the receiver whose class matches *code*.

Discussion

`SBObject` subclasses use this method to implement application-specific property accessor methods. You should not need to call this method directly.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- propertyWithCode:` (page 32)

Declared In

`SBObject.h`

get

Forces evaluation of the receiver, causing the real object to be returned immediately.

```
- (id)get
```

Return Value

The object referenced by the receiver.

Discussion

This method forces the current object reference (the receiver) to be evaluated, resulting in the return of the referenced object. By default, Scripting Bridge deals with references to objects until you actually request some concrete data from them or until you call the `get` method.

Availability

Available in Mac OS X v10.5 and later.

Declared In

`SBObject.h`

init

Initializes and returns an instance of an `SBObject` subclass.

```
- (id)init
```

Return Value

An `SBObject` object or `nil` if the object could not be initialized.

Discussion

Scripting Bridge does not actually create an object in the target application until you add the object returned from this method to an element array (`SBElementArray`).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `initWithProperties:` (page 31)
- `initWithData:` (page 30)
- `initWithElementCode:properties:data:` (page 30)

Declared In
`SBObject.h`

initWithData:

Returns an instance of an `SBObject` subclass initialized with the given data.

```
- (id)initWithData:(id)data
```

Parameters

data

An object containing data for the new `SBObject` object. The data varies according to the type of scripting object to be created.

Return Value
An `SBObject` object or `nil` if the object could not be initialized.

Discussion
Scripting Bridge does not actually create an object in the target application until you add the object returned from this method to an element array (`SBElementArray`).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `init` (page 29)
- `initWithProperties:` (page 31)
- `initWithElementCode:properties:data:` (page 30)

Declared In
`SBObject.h`

initWithElementCode:properties:data:

Returns an instance of an `SBObject` subclass initialized with the specified properties and data and added to the designated element array.

```
- (id)initWithElementCode:(DescType)code properties:(NSDictionary *)properties
    data:(id)data
```

Parameters

code

A four-character code used to identify an element in the target application's scripting interface. See *Apple Event Manager Reference* for details.

properties

A dictionary with keys specifying the names of properties (that is, attributes or to-one relationships) and the values for those properties. Pass `nil` if you are initializing the object by *data* only.

data

An object containing data for the new `SBObject` object. The data varies according to the type of scripting object to be created. Pass `nil` if you initializing the object by *properties* only.

Return Value
An `SBObject` object or `nil` if the object could not be initialized.

Discussion
Unlike the other initializers of this class, this method not only initializes the `SBObject` object but adds it to a specified element array. This method is the designated initializer.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `init` (page 29)
- `initWithData:` (page 30)
- `initWithProperties:` (page 31)

Declared In
`SBObject.h`

initWithProperties:

Returns an instance of an `SBObject` subclass initialized with the specified properties.

```
- (id)initWithProperties:(NSDictionary *)properties
```

Parameters

properties

A dictionary with keys specifying the names of properties (that is, attributes or to-one relationships) and the values for those properties.

Return Value
An `SBObject` object or `nil` if the object could not be initialized.

Discussion
Scripting Bridge does not actually create an object in the target application until you add the object returned from this method to an element array (`SBElementArray`).

Availability
Available in Mac OS X v10.5 and later.

See Also
- `init` (page 29)
- `initWithData:` (page 30)
- `initWithElementCode:properties:data:` (page 30)

Declared In
`SBObject.h`

propertyWithClass:code:

Returns an object of the designated scripting class representing the specified property of the receiver

```
- (SBObject *)propertyWithClass:(Class)class code:(AEKeyword)code
```

Parameters

class

The `SBObject` subclass with which to instantiate the object.

code

A four-character code that uniquely identifies a property of the receiver.

Return Value
An instance of the designated *class* that represents the receiver's property identified by *code*.

Discussion
`SBObject` subclasses use this method to implement application-specific property accessor methods. You should not need to call this method directly.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `propertyWithCode:` (page 32)

Declared In
`SBObject.h`

propertyWithCode:

Returns an object representing the specified property of the receiver.

```
- (SBObject *)propertyWithCode:(AEKeyword)code
```

Parameters

code

A four-character code that uniquely identifies a property of the receiver.

Return Value
An object representing the receiver's property as identified by *code*.

Discussion
`SBObject` subclasses use this method to implement application-specific property accessor methods. You should not need to call this method directly.

Availability
Available in Mac OS X v10.5 and later.

See Also
- `propertyWithClass:code:` (page 32)
- `elementArrayWithCode:` (page 28)

Declared In
`SBObject.h`

sendEvent:id:parameters:

Sends an Apple event with the given event class, event ID, and format to the target application.

```
- (id)sendEvent:(AEEventClass)eventClass id:(AEEventID)eventID
    parameters:(DescType)firstParamCode,...
```

Parameters

eventClass

The event class of the Apple event to be sent.

eventID

The event ID of the Apple event to be sent.

firstParamCode,...

A list of four-character parameter codes (`DescType`) and object values (`id`) terminated by a zero.

Return Value

The target application's Apple event sent in reply; it is converted to a Cocoa object of an appropriate type.

Discussion

Scripting Bridge uses this method to communicate with target applications. If the target application responds to this method by sending an Apple event representing an error, the receiver calls its delegate's `eventDidFail:withError:` (page 37) method. If no delegate has been assigned, the receiver raises an exception.

You should rarely have to call this method directly.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- setTo:` (page 33)

Declared In

`SBObject.h`

setTo:

Sets the receiver to a specified value.

```
- (void)setTo:(id)value
```

Parameters

value

The data the receiver should be set to. It can be an `NSString`, `NSNumber`, `NSArray`, `SBObject`, or any other type of object supported by the Scripting Bridge framework.

Discussion

You should not call this method directly.

Availability

Available in Mac OS X v10.5 and later.

See Also

`- sendEvent:id:parameters:` (page 33)

Declared In
`SBObject.h`

Protocols

SBApplicationDelegate Protocol Reference

Framework	/System/Library/Frameworks/ScriptingBridge.framework
Availability	Available in Mac OS X v10.6 and later.
Declared in	ScriptingBridge/SBApplication.h

Overview

This informal protocol defines a delegation method for handling Apple event errors that are sent from an target application to an `SBApplication` object.

You must set a delegate for the `SBApplication` object using the `setDelegate:` (page 18) method. If you do not set a delegate and have the delegate handle the error in some way, `SBApplication` raises an exception.

Tasks

Handling Errors

`- eventDidFail:withError:` (page 37) *required method*

Sent by an `SBApplication` object when a target application returns an error Apple event. (required)

Instance Methods

eventDidFail:withError:

Sent by an `SBApplication` object when a target application returns an error Apple event. (required)

```
- (void)eventDidFail:(const AppleEvent *)event withError:(NSError *)error
```

Parameters

event

A pointer to the Apple event sent to the target application causing the error.

error

An object containing information about the error Apple event. Specific information may be included in the `useInfo` dictionary of the error object. See "User Info Dictionary Keys" (page 38) for a list of possible keys for this dictionary.

Availability
Available in Mac OS X v10.5 and later.
Available as part of an informal protocol prior to Mac OS X v10.6.

Declared In
`SBApplication.h`

Constants

User Info Dictionary Keys

The following describes the possible keys for the `userInfo` dictionary of the `NSError` object passed to the delegate. Note that for some errors, the `userInfo` dictionary may not have any of these keys.

Constants

`@"ErrorBriefMessage"`
A short human-readble description of the error, as an `NSString` object.

`@"ErrorExpectedType"`
The type of data the target application expected, as an `NSAppleEventDescriptor` object.

`@"ErrorOffendingObject"`
The object that caused the error.

`@"ErrorString"`
A full human-readable description of the error, as an `NSString` object.

`@"ErrorNumber"`
The Apple event error number, as an `NSNumber` object.

Declared In
`ScriptingBridge/SBApplication.h`

Document Revision History

This table describes the changes to *Scripting Bridge Framework Reference*.

Date	Notes
2007-05-29	A new document that describes the Objective-C API that allows Cocoa applications to communicate with scriptable applications.

Document Revision History

Index

T

U