NSObject Class Reference

Developer

Contents

NSObject Class Reference

Inherits from	none (NSObject is a root class)
Conforms to	NSObject
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in OS X v10.0 and later.
Declared in	NSArchiver.h NSClassDescription.h NSKeyedArchiver.h NSObject.h NSObjectScripting.h NSPortCoder.h NSRunLoop.h NSScriptClassDescription.h NSThread.h objc/NSObject.h
Related sample code	Denoise QTCoreVideo102 QTCoreVideo103 QTCoreVideo201 QTCoreVideo202

Overview

`NSObject` is the root class of most Objective-C class hierarchies. Through `NSObject`, objects inherit a basic interface to the runtime system and the ability to behave as Objective-C objects.

Tasks

Initializing a Class

`+ initialize` (page 19)

Initializes the class before it receives its first message.

`+ load` (page 23)

Invoked whenever a class or category is added to the Objective-C runtime; implement this method to perform class-specific behavior upon loading.

Creating, Copying, and Deallocating Objects

`+ alloc` (page 11)

Returns a new instance of the receiving class.

`+ allocWithZone:` (page 12)

Returns a new instance of the receiving class.

`– init` (page 43)

Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

`– copy` (page 35)

Returns the object returned by `copyWithZone:`.

`+ copyWithZone:` (page 18)

Returns the receiver.

`– mutableCopy` (page 46)

Returns the object returned by `mutableCopyWithZone:` where the zone is `nil`.

`+ mutableCopyWithZone:` (page 24)

Returns the receiver.

`– dealloc` (page 37)

Deallocates the memory occupied by the receiver.

`+ new` (page 24)

Allocates a new instance of the receiving class, sends it an `init` (page 43) message, and returns the initialized object.

Identifying Classes

`+ class` (page 15)

Returns the class object.

`+ superclass` (page 28)

Returns the class object for the receiver's superclass.

`+ isSubclassOfClass:` (page 22)

Returns a Boolean value that indicates whether the receiving class is a subclass of, or identical to, a given class.

Testing Class Functionality

`+ instancesRespondToSelector:` (page 22)

Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

Testing Protocol Conformance

`+ conformsToProtocol:` (page 17)

Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

Obtaining Information About Methods

`- methodForSelector:` (page 45)

Locates and returns the address of the receiver's implementation of a method so it can be called as a function.

`+ instanceMethodForSelector:` (page 20)

Locates and returns the address of the implementation of the instance method identified by a given selector.

`+ instanceMethodSignatureForSelector:` (page 21)

Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.

`- methodSignatureForSelector:` (page 45)

Returns an `NSMethodSignature` object that contains a description of the method identified by a given selector.

Describing Objects

+ `description` (page 19)

Returns a string that represents the contents of the receiving class.

Discardable Content Proxy Support

– `autoContentAccessingProxy` (page 30)

Creates and returns a proxy for the receiving object

Sending Messages

– `performSelector:withObject:afterDelay:` (page 51)

Invokes a method of the receiver on the current thread using the default mode after a delay.

– `performSelector:withObject:afterDelay:inModes:` (page 52)

Invokes a method of the receiver on the current thread using the specified modes after a delay.

– `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54)

Invokes a method of the receiver on the main thread using the default mode.

– `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56)

Invokes a method of the receiver on the main thread using the specified modes.

– `performSelector:onThread:withObject:waitUntilDone:` (page 48)

Invokes a method of the receiver on the specified thread using the default mode.

– `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

Invokes a method of the receiver on the specified thread using the specified modes.

– `performSelectorInBackground:withObject:` (page 53)

Invokes a method of the receiver on a new background thread.

+ `cancelPreviousPerformRequestsWithTarget:` (page 13)

Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method.

+ `cancelPreviousPerformRequestsWithTarget:selector:object:` (page 14)

Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 51).

Forwarding Messages

– `forwardingTargetForSelector:` (page 40)
Returns the object to which unrecognized messages should first be directed.

– `forwardInvocation:` (page 41)
Overridden by subclasses to forward messages to other objects.

Dynamically Resolving Methods

+ `resolveClassMethod:` (page 25)
Dynamically provides an implementation for a given selector for a class method.

+ `resolveInstanceMethod:` (page 26)
Dynamically provides an implementation for a given selector for an instance method.

Error Handling

– `doesNotRecognizeSelector:` (page 38)
Handles messages the receiver doesn't recognize.

Archiving

– `awakeAfterUsingCoder:` (page 30)
Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.

– `classForArchiver` (page 32)
Overridden by subclasses to substitute a class other than its own during archiving.

– `classForCoder` (page 33)
Overridden by subclasses to substitute a class other than its own during coding.

– `classForKeyedArchiver` (page 33)
Overridden by subclasses to substitute a new class for instances during keyed archiving.

+ `classFallbacksForKeyedArchiver` (page 16)
Overridden to return the names of classes that can be used to decode objects if their class is unavailable.

+ `classForKeyedUnarchiver` (page 16)
Overridden by subclasses to substitute a new class during keyed unarchiving.

– `classForPortCoder` (page 34)
Overridden by subclasses to substitute a class other than its own for distribution encoding.

– `replacementObjectForArchiver:` (page 57)
Overridden by subclasses to substitute another object for itself during archiving.

– `replacementObjectForCoder:` (page 58)
Overridden by subclasses to substitute another object for itself during encoding.

– `replacementObjectForKeyedArchiver:` (page 58)
Overridden by subclasses to substitute another object for itself during keyed archiving.

– `replacementObjectForPortCoder:` (page 59)
Overridden by subclasses to substitute another object or a copy for itself during distribution encoding.

+ `setVersion:` (page 27)
Sets the receiver's version number.

+ `version` (page 28)
Returns the version number assigned to the class.

Working with Class Descriptions

– `attributeKeys` (page 29)
Returns an array of `NSString` objects containing the names of immutable values that instances of the receiver's class contain.

– `classDescription` (page 32)
Returns an object containing information about the attributes and relationships of the receiver’s class.

– `inverseForRelationshipKey:` (page 44)
For a given key that defines the name of the relationship from the receiver’s class to another class, returns the name of the relationship from the other class to the receiver’s class.

– `toManyRelationshipKeys` (page 62)
Returns array containing the keys for the to-many relationship properties of the receiver.

– `toOneRelationshipKeys` (page 63)
Returns the keys for the to-one relationship properties of the receiver, if any.

Scripting

– `classCode` (page 31)
Returns the receiver's Apple event type code, as stored in the `NSScriptClassDescription` object for the object's class.

– `className` (page 34)
Returns a string containing the name of the class.

– `copyScriptingValue:forKey:withProperties:` (page 36)
Creates and returns one or more scripting objects to be inserted into the specified relationship by copying the passed-in value and setting the properties in the copied object or objects.

– `newScriptingObjectOfClass:forValueForKey:withContentsValue:properties:` (page 47)
Creates and returns an instance of a scriptable class, setting its contents and properties, for insertion into the relationship identified by the key.

– `scriptingProperties` (page 60)
Returns an `NSString`-keyed dictionary of the receiver's scriptable properties.

– `setScriptingProperties:` (page 61)
Given an `NSString`-keyed dictionary, sets one or more scriptable properties of the receiver.

– `scriptingValueForSpecifier:` (page 60)
Given an object specifier, returns the specified object or objects in the receiving container.

Deprecated Methods

– `finalize` (page 39)
The garbage collector invokes this method on the receiver before disposing of the memory it uses. (Deprecated. Garbage collection is deprecated in OS X v10.8; instead, you should use Automatic Reference Counting—see *Transitioning to ARC Release Notes* .)

+ `poseAsClass:` (page 64) Deprecated in OS X v10.5
Causes the receiving class to pose as a specified superclass.

Class Methods

alloc

Returns a new instance of the receiving class.

```
+ (id)alloc
```

Return Value
A new instance of the receiver.

Discussion
The `isa` instance variable of the new instance is initialized to a data structure that describes the class; memory for all other instance variables is set to `0`.

You must use an `init...` method to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass alloc] init];
```

Do not override `alloc` to include initialization code. Instead, implement class-specific versions of `init...` methods.

For historical reasons, `alloc` invokes `allocWithZone:`.

Availability
Available in OS X v10.0 and later.

See Also
– `init` (page 43)

Related Sample Code
From A View to A Movie
GLSL Showpiece Lite
QTCoreVideo102
QTCoreVideo201
QTCoreVideo202

Declared in
`objc/NSObject.h`

allocWithZone:

Returns a new instance of the receiving class.

```
+ (id)allocWithZone:(NSZone *)zone
```

Parameters
`zone`

This parameter is ignored.

Return Value
A new instance of the receiver.

Discussion
The `isa` instance variable of the new instance is initialized to a data structure that describes the class; memory for all other instance variables is set to `0`.

You must use an `init...` method to complete the initialization process. For example:

```
TheClass *newObject = [[TheClass allocWithZone:nil] init];
```

Do not override `allocWithZone:` to include any initialization code. Instead, class-specific versions of `init...` methods.

This method exists for historical reasons; memory zones are no longer used by Objective-C.

Availability
Available in OS X v10.0 and later.

See Also
`+ alloc` (page 11)
`– init` (page 43)

Related Sample Code
From A View to A Movie
From A View to A Picture
MenuItemView
MenuMadness
QTCoreVideo301

Declared in
`objc/NSObject.h`

cancelPreviousPerformRequestsWithTarget:

Cancels perform requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method.

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget
```

Parameters

`aTarget`

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method.

Discussion

All perform requests having the same target `aTarget` are canceled. This method removes perform requests only in the current run loop, not all run loops.

Availability

Available in OS X v10.2 and later.

Related Sample Code

FunkyOverlayWindow

Declared in

`NSRunLoop.h`

cancelPreviousPerformRequestsWithTarget:selector:object:

Cancels perform requests previously registered with `performSelector:withObject:afterDelay:` (page 51).

```
+ (void)cancelPreviousPerformRequestsWithTarget:(id)aTarget selector:(SEL)aSelector
 object:(id)anArgument
```

Parameters

`aTarget`

The target for requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method

`aSelector`

The selector for requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method.

`anArgument`

The argument for requests previously registered with the `performSelector:withObject:afterDelay:` (page 51) instance method. Argument equality is determined using `isEqual:`, so the value need not be the same object that was passed originally. Pass `nil` to match a request for `nil` that was originally passed as the argument.

Discussion

All perform requests are canceled that have the same target as `aTarget`, argument as `anArgument`, and selector as `aSelector`. This method removes perform requests only in the current run loop, not all run loops.

Availability
Available in OS X v10.0 and later.

Related Sample Code
AttachAScript

Declared in
`NSRunLoop.h`

class

Returns the class object.

```
+ (Class)class
```

Return Value
The class object.

Discussion
Refer to a class only by its name when it is the receiver of a message. In all other cases, the class object must be obtained through this or a similar method. For example, here `SomeClass` is passed as an argument to the `isKindOfClass:` method (declared in the `NSObject` protocol):

```
BOOL test = [self isKindOfClass:[SomeClass class]];
```

Availability
Available in OS X v10.0 and later.

See Also
`class` (`NSObject` protocol)
`NSClassFromString`
`NSStringFromClass`

Related Sample Code
GLSLShowpiece
Sketch
Sketch+Accessibility
TableViewPlayground
TextEdit

Declared in
`objc/NSObject.h`

classFallbacksForKeyedArchiver

Overridden to return the names of classes that can be used to decode objects if their class is unavailable.

```
+ (NSArray *)classFallbacksForKeyedArchiver
```

Return Value
An array of `NSString` objects that specify the names of classes in preferred order for unarchiving

Discussion
`NSKeyedArchiver` calls this method and stores the result inside the archive. If the actual class of an object doesn't exist at the time of unarchiving, `NSKeyedUnarchiver` goes through the stored list of classes and uses the first one that does exists as a substitute class for decoding the object. The default implementation of this method returns `nil`.

You can use this method if you introduce a new class into your application to provide some backwards compatibility in case the archive will be read on a system that does not have that class. Sometimes there may be another class which may work nearly as well as a substitute for the new class, and the archive keys and archived state for the new class can be carefully chosen (or compatibility written out) so that the object can be unarchived as the substitute class if necessary.

Availability
Available in OS X v10.4 and later.

Declared in
`NSKeyedArchiver.h`

classForKeyedUnarchiver

Overridden by subclasses to substitute a new class during keyed unarchiving.

```
+ (Class)classForKeyedUnarchiver
```

Return Value
The class to substitute for the receiver during keyed unarchiving.

Discussion
During keyed unarchiving, instances of the receiver will be decoded as members of the returned class. This method overrides the results of the decoder's class and instance name to class encoding tables.

Availability
Available in OS X v10.2 and later.

Declared in
NSKeyedArchiver.h

conformsToProtocol:

Returns a Boolean value that indicates whether the receiver conforms to a given protocol.

```
+ (BOOL)conformsToProtocol:(Protocol *)aProtocol
```

Parameters
aProtocol
A protocol.

Return Value
YES if the receiver conforms to aProtocol, otherwise NO.

Discussion
A class is said to "conform to" a protocol if it adopts the protocol or inherits from another class that adopts it. Protocols are adopted by listing them within angle brackets after the interface declaration. For example, here MyClass adopts the (fictitious) AffiliationRequests and Normalization protocols:

```
@interface MyClass : NSObject <AffiliationRequests, Normalization>
```

A class also conforms to any protocols that are incorporated in the protocols it adopts or inherits. Protocols incorporate other protocols in the same way classes adopt them. For example, here the AffiliationRequests protocol incorporates the Joining protocol:

```
@protocol AffiliationRequests <Joining>
```

If a class adopts a protocol that incorporates another protocol, it must also implement all the methods in the incorporated protocol or inherit those methods from a class that adopts it.

This method determines conformance solely on the basis of the formal declarations in header files, as illustrated above. It doesn't check to see whether the methods declared in the protocol are actually implemented—that's the programmer's responsibility.

The protocol required as this method's argument can be specified using the @protocol() directive:

```
BOOL canJoin = [MyClass conformsToProtocol:@protocol(Joining)];
```

Availability
Available in OS X v10.0 and later.

See Also
`+ conformsToProtocol:` (page 17)

Declared in
`objc/NSObject.h`

copyWithZone:

Returns the receiver.

```
+ (id)copyWithZone:(NSZone *)zone
```

Parameters
`zone`

This argument is ignored.

Return Value
The receiver.

Discussion
This method exists so class objects can be used in situations where you need an object that conforms to the `NSCopying` protocol. For example, this method lets you use a class object as a key to an `NSDictionary` object. You should not override this method.

Availability
Available in OS X v10.0 and later.

See Also
`– copy` (page 35)

Related Sample Code
BezierPathLab

CompositeLab

TextEdit

Declared in
`objc/NSObject.h`

description

Returns a string that represents the contents of the receiving class.

```
+ (NSString *)description
```

Return Value
A string that represents the contents of the receiving class.

Discussion
The debugger's print-object command invokes this method to produce a textual description of an object.

`NSObject`'s implementation of this method simply prints the name of the class.

Availability
Available in OS X v10.0 and later.

See Also
`description` (`NSObject` protocol)

Related Sample Code
AVCompositionDebugViewer
AVRecorder
QTRecorder
Sketch+Accessibility
TreeView

Declared in
`objc/NSObject.h`

initialize

Initializes the class before it receives its first message.

```
+ (void)initialize
```

Discussion
The runtime sends `initialize` to each class in a program just before the class, or any class that inherits from it, is sent its first message from within the program. The runtime sends the `initialize` message to classes in a thread-safe manner. Superclasses receive this message before their subclasses. The superclass implementation may be called multiple times if subclasses do not implement `initialize`—the runtime will call the inherited implementation—or if subclasses explicitly call `[super initialize]`. If you want to protect yourself from being run multiple times, you can structure your implementation along these lines:

```
+ (void)initialize {
  if (self == [ClassName self]) {
    // ... do the initialization ...
  }
}
```

Because `initialize` is called in a thread-safe manner and the order of `initialize` being called on different classes is not guaranteed, it's important to do the minimum amount of work necessary in `initialize` methods. Specifically, any code that takes locks that might be required by other classes in their `initialize` methods is liable to lead to deadlocks. Therefore you should not rely on `initialize` for complex initialization, and should instead limit it to straightforward, class local initialization.

Special Considerations

`initialize` is invoked only once per class. If you want to perform independent initialization for the class and for categories of the class, you should implement `load` (page 23) methods.

Availability

Available in OS X v10.0 and later.

See Also

– `init` (page 43)
+ `load` (page 23)
`class` (`NSObject` protocol)

Related Sample Code

Cache
CameraBrowser
Dicey
QuickLookSketch
Scene Kit Material Editor

Declared in

`objc/NSObject.h`

instanceMethodForSelector:

Locates and returns the address of the implementation of the instance method identified by a given selector.

```
+ (IMP)instanceMethodForSelector:(SEL)aSelector
```

Parameters

`aSelector`

A selector that identifies the method for which to return the implementation address. The selector must be non-`NULL` and valid for the receiver. If in doubt, use the `respondsToSelector:` method to check before passing the selector to `methodForSelector:`.

Return Value

The address of the implementation of the `aSelector` instance method.

Discussion

An error is generated if instances of the receiver can't respond to `aSelector` messages.

Use this method to ask the class object for the implementation of instance methods only. To ask the class for the implementation of a class method, send the `methodForSelector:` (page 45) instance method to the class instead.

Availability

Available in OS X v10.0 and later.

Declared in

`objc/NSObject.h`

instanceMethodSignatureForSelector:

Returns an `NSMethodSignature` object that contains a description of the instance method identified by a given selector.

```
+ (NSMethodSignature *)instanceMethodSignatureForSelector:(SEL)aSelector
```

Parameters

`aSelector`

A selector that identifies the method for which to return the implementation address.

Return Value

An `NSMethodSignature` object that contains a description of the instance method identified by `aSelector`, or `nil` if the method can't be found.

Availability

Available in OS X v10.0 and later.

See Also

– `methodSignatureForSelector:` (page 45)

Declared in
objc/NSObject.h

instancesRespondToSelector:

Returns a Boolean value that indicates whether instances of the receiver are capable of responding to a given selector.

```
+ (BOOL)instancesRespondToSelector:(SEL)aSelector
```

Parameters
aSelector
A selector.

Return Value
YES if instances of the receiver are capable of responding to aSelector messages, otherwise NO.

Discussion
If aSelector messages are forwarded to other objects, instances of the class are able to receive those messages without error even though this method returns NO.

To ask the class whether it, rather than its instances, can respond to a particular message, send to the class instead the NSObject protocol instance method respondsToSelector:.

Availability
Available in OS X v10.0 and later.

See Also
– forwardInvocation: (page 41)

Declared in
objc/NSObject.h

isSubclassOfClass:

Returns a Boolean value that indicates whether the receiving class is a subclass of, or identical to, a given class.

```
+ (BOOL)isSubclassOfClass:(Class)aClass
```

Parameters
aClass
A class object.

Return Value
`YES` if the receiving class is a subclass of—or identical to—`aClass`, otherwise `NO`.

Availability
Available in OS X v10.2 and later.

Declared in
`objc/NSObject.h`

load

Invoked whenever a class or category is added to the Objective-C runtime; implement this method to perform class-specific behavior upon loading.

```
+ (void)load
```

Discussion
The `load` message is sent to classes and categories that are both dynamically loaded and statically linked, but only if the newly loaded class or category implements a method that can respond.

The order of initialization is as follows:

1. All initializers in any framework you link to.
2. All `+load` methods in your image.
3. All C++ static initializers and C/C++ `__attribute__(constructor)` functions in your image.
4. All initializers in frameworks that link to you.

In addition:

- A class's `+load` method is called after all of its superclasses' `+load` methods.
- A category `+load` method is called after the class's own `+load` method.

In a custom implementation of `load` you can therefore safely message other unrelated classes from the same image, but any `load` methods implemented by those classes may not have run yet.

Availability
Available in OS X v10.0 and later.

See Also
`+ initialize` (page 19)

Related Sample Code
CIAnnotation

CIDemoImageUnit

Declared in
`objc/NSObject.h`

mutableCopyWithZone:

Returns the receiver.

```
+ (id)mutableCopyWithZone:(NSZone *)zone
```

Parameters
`zone`

The memory zone in which to create the copy of the receiver.

Return Value
The receiver.

Discussion
This method exists so class objects can be used in situations where you need an object that conforms to the `NSMutableCopying` protocol. For example, this method lets you use a class object as a key to an `NSDictionary` object. You should not override this method.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/NSObject.h`

new

Allocates a new instance of the receiving class, sends it an `init` (page 43) message, and returns the initialized object.

```
+ (id)new
```

Return Value
A new instance of the receiver.

Discussion

This method is a combination of `alloc` (page 11) and `init` (page 43). Like `alloc` (page 11), it initializes the `isa` instance variable of the new object so it points to the class data structure. It then invokes the `init` (page 43) method to complete the initialization process.

Availability

Available in OS X v10.0 and later.

Related Sample Code

From A View to A Movie

QTCoreVideo102

QTCoreVideo103

QTCoreVideo201

QTCoreVideo202

Declared in

`objc/NSObject.h`

resolveClassMethod:

Dynamically provides an implementation for a given selector for a class method.

```
+ (BOOL)resolveClassMethod:(SEL)name
```

Parameters

`name`

The name of a selector to resolve.

Return Value

`YES` if the method was found and added to the receiver, otherwise `NO`.

Discussion

This method allows you to dynamically provide an implementation for a given selector. See `resolveInstanceMethod:` (page 26) for further discussion.

Availability

Available in OS X v10.5 and later.

See Also

`+ resolveInstanceMethod:` (page 26)

Declared in

`objc/NSObject.h`

resolveInstanceMethod:

Dynamically provides an implementation for a given selector for an instance method.

```
+ (BOOL)resolveInstanceMethod:(SEL)name
```

Parameters

`name`

The name of a selector to resolve.

Return Value

`YES` if the method was found and added to the receiver, otherwise `NO`.

Discussion

This method and `resolveClassMethod:` (page 25) allow you to dynamically provide an implementation for a given selector.

An Objective-C method is simply a C function that take at least two arguments—`self` and `_cmd`. Using the `class_addMethod` function, you can add a function to a class as a method. Given the following function:

```
void dynamicMethodIMP(id self, SEL _cmd)
{
    // implementation ....
}
```

you can use `resolveInstanceMethod:` to dynamically add it to a class as a method (called `resolveThisMethodDynamically`) like this:

```
+ (BOOL) resolveInstanceMethod:(SEL)aSEL
{
    if (aSEL == @selector(resolveThisMethodDynamically))
    {
          class_addMethod([self class], aSEL, (IMP) dynamicMethodIMP, "v@:");
          return YES;
    }
    return [super resolveInstanceMethod:aSel];
}
```

Special Considerations

This method is called before the Objective-C forwarding mechanism is invoked. If `respondsToSelector:` or `instancesRespondToSelector:` (page 22) is invoked, the dynamic method resolver is given the opportunity to provide an `IMP` for the given selector first.

Availability

Available in OS X v10.5 and later.

See Also

`+ resolveClassMethod:` (page 25)

Declared in

`objc/NSObject.h`

setVersion:

Sets the receiver's version number.

```
+ (void)setVersion:(NSInteger)aVersion
```

Parameters

`aVersion`

The version number for the receiver.

Discussion

The version number is helpful when instances of the class are to be archived and reused later. The default version is `0`.

Special Considerations

The version number applies to `NSArchiver`/`NSUnarchiver`, but not to `NSKeyedArchiver`/`NSKeyedUnarchiver`. A keyed archiver does not encode class version numbers.

Availability

Available in OS X v10.0 and later.

See Also

`+ version` (page 28)

Declared in

`NSObject.h`

superclass

Returns the class object for the receiver's superclass.

```
+ (Class)superclass
```

Return Value
The class object for the receiver's superclass.

Availability
Available in OS X v10.0 and later.

See Also
`+ class` (page 15)
`superclass` (`NSObject` protocol)

Declared in
`objc/NSObject.h`

version

Returns the version number assigned to the class.

```
+ (NSInteger)version
```

Return Value
The version number assigned to the class.

Discussion
If no version has been set, the default is `0`.

Version numbers are needed for decoding or unarchiving, so older versions of an object can be detected and decoded correctly.

Caution should be taken when obtaining the version from within an `NSCoding` protocol or other methods. Use the class name explicitly when getting a class version number:

```
version = [MyClass version];
```

Don't simply send `version` to the return value of class—a subclass version number may be returned instead.

Special Considerations

The version number applies to `NSArchiver`/`NSUnarchiver`, but not to `NSKeyedArchiver`/`NSKeyedUnarchiver`. A keyed archiver does not encode class version numbers.

Availability

Available in OS X v10.0 and later.

See Also

`+ setVersion:` (page 27)
`versionForClassName:` (`NSCoder`)

Related Sample Code

GLEssentials

Declared in

`NSObject.h`

Instance Methods

attributeKeys

Returns an array of `NSString` objects containing the names of immutable values that instances of the receiver's class contain.

```
- (NSArray *)attributeKeys
```

Return Value

An array of `NSString` objects containing the names of immutable values that instances of the receiver's class contain.

Discussion

`NSObject`'s implementation of `attributeKeys` simply calls `[[self classDescription] attributeKeys]`. To make use of the default implementation, you must therefore implement and register a suitable class description—see `NSClassDescription`. A class description that describes Movie objects could, for example, return the attribute keys `title`, `dateReleased`, and `rating`.

Availability

Available in OS X v10.0 and later.

See Also

`- classDescription` (page 32)
`- inverseForRelationshipKey:` (page 44)

– `toManyRelationshipKeys` (page 62)
– `toOneRelationshipKeys` (page 63)

Declared in
`NSClassDescription.h`

autoContentAccessingProxy

Creates and returns a proxy for the receiving object

```
– (id)autoContentAccessingProxy
```

Return Value
A proxy of the receiver.

Discussion
This method creates and returns a proxy for the receiving object if the receiver adopts the `NSDiscardableContent` protocol and still has content that has not been discarded.

The proxy calls `beginContentAccess` on the receiver to keep the content available as long as the proxy lives, and calls `endContentAccess` when the proxy is deallocated.

The wrapper object is otherwise a subclass of `NSProxy` and forwards messages to the original receiver object as an NSProxy does.

This method can be used to hide an `NSDiscardableContent` object's content volatility by creating an object that responds to the same messages but holds the contents of the original receiver available as long as the created proxy lives. Thus hidden, the `NSDiscardableContent` object (by way of the proxy) can be given out to unsuspecting recipients of the object who would otherwise not know they might have to call `beginContentAccess` and `endContentAccess` around particular usages (specific to each `NSDiscardableContent` object) of the `NSDiscardableContent` object.

Availability
Available in OS X v10.6 and later.

Declared in
`NSObject.h`

awakeAfterUsingCoder:

Overridden by subclasses to substitute another object in place of the object that was decoded and subsequently received this message.

```
- (id)awakeAfterUsingCoder:(NSCoder *)aDecoder
```

Parameters

`aDecoder`

The decoder used to decode the receiver.

Return Value

The receiver, or another object to take the place of the object that was decoded and subsequently received this message.

Discussion

You can use this method to eliminate redundant objects created by the coder. For example, if after decoding an object you discover that an equivalent object already exists, you can return the existing object. If a replacement is returned, your overriding method is responsible for releasing the receiver.

This method is invoked by `NSCoder`. `NSObject`'s implementation simply returns `self`.

Availability

Available in OS X v10.0 and later.

See Also

`– classForCoder` (page 33)
`– replacementObjectForCoder:` (page 58)
`initWithCoder:` (`NSCoding` protocol)

Declared in

`NSObject.h`

classCode

Returns the receiver's Apple event type code, as stored in the `NSScriptClassDescription` object for the object's class.

```
- (FourCharCode)classCode
```

Return Value

The receiver's Apple event type code, as stored in the `NSScriptClassDescription` object for the object's class.

Discussion

This method is invoked by Cocoa's scripting support classes.

Availability
Available in OS X v10.0 and later.

Declared in
`NSScriptClassDescription.h`

classDescription

Returns an object containing information about the attributes and relationships of the receiver's class.

```
- (NSClassDescription *)classDescription
```

Return Value
An object containing information about the attributes and relationships of the receiver's class.

Discussion
`NSObject`'s implementation simply calls `[NSClassDescription classDescriptionForClass:[self class]]`. See `NSClassDescription` for more information.

Availability
Available in OS X v10.0 and later.

See Also
- `attributeKeys` (page 29)
- `inverseForRelationshipKey:` (page 44)
- `toManyRelationshipKeys` (page 62)
- `toOneRelationshipKeys` (page 63)

Related Sample Code
SimpleScriptingPlugin

Declared in
`NSClassDescription.h`

classForArchiver

Overridden by subclasses to substitute a class other than its own during archiving.

```
- (Class)classForArchiver
```

Return Value
The class to substitute for the receiver's own class during archiving.

Discussion

This method is invoked by `NSArchiver`. It allows specialized behavior for archiving—for example, the private subclasses of a class cluster substitute the name of their public superclass when being archived.

`NSObject`'s implementation returns the object returned by `classForCoder` (page 33). Override `classForCoder` (page 33) to add general coding behavior.

Availability

Available in OS X v10.0 and later.

See Also

– `replacementObjectForArchiver:` (page 57)

Declared in

`NSArchiver.h`

classForCoder

Overridden by subclasses to substitute a class other than its own during coding.

```
- (Class)classForCoder
```

Return Value

The class to substitute for the receiver's own class during coding.

Discussion

This method is invoked by `NSCoder`. `NSObject`'s implementation returns the receiver's class. The private subclasses of a class cluster substitute the name of their public superclass when being archived.

Availability

Available in OS X v10.0 and later.

See Also

– `awakeAfterUsingCoder:` (page 30)
– `replacementObjectForCoder:` (page 58)

Declared in

`NSObject.h`

classForKeyedArchiver

Overridden by subclasses to substitute a new class for instances during keyed archiving.

```
- (Class)classForKeyedArchiver
```

Discussion
The object will be encoded as if it were a member of the returned class. The results of this method are overridden by the encoder class and instance name to class encoding tables. If `nil` is returned, the result of this method is ignored.

Availability
Available in OS X v10.2 and later.

See Also
`– replacementObjectForKeyedArchiver:` (page 58)

Declared in
`NSKeyedArchiver.h`

classForPortCoder

Overridden by subclasses to substitute a class other than its own for distribution encoding.

```
- (Class)classForPortCoder
```

Return Value
The class to substitute for the receiver in distribution encoding.

Discussion
This method allows specialized behavior for distributed objects—override `classForCoder` (page 33) to add general coding behavior. This method is invoked by `NSPortCoder`. `NSObject`'s implementation returns the class returned by `classForCoder` (page 33).

Availability
Available in OS X v10.0 and later.

See Also
`– replacementObjectForPortCoder:` (page 59)

Declared in
`NSPortCoder.h`

className

Returns a string containing the name of the class.

```
- (NSString *)className
```

Return Value
A string containing the name of the class.

Discussion
This method is invoked by Cocoa's scripting support classes.

Availability
Available in OS X v10.0 and later.

Related Sample Code
DragNDropOutlineView
QuickLookSketch
Sketch
Sketch+Accessibility

Declared in
`NSScriptClassDescription.h`

copy

Returns the object returned by `copyWithZone:`.

```
- (id)copy
```

Return Value
The object returned by the `NSCopying` protocol method `copyWithZone:`,.

Discussion
This is a convenience method for classes that adopt the `NSCopying` protocol. An exception is raised if there is no implementation for `copyWithZone:`.

`NSObject` does not itself support the `NSCopying` protocol. Subclasses must support the protocol and implement the `copyWithZone:` method. A subclass version of the `copyWithZone:` method should send the message to `super` first, to incorporate its implementation, unless the subclass descends directly from `NSObject`.

Availability
Available in OS X v10.0 and later.

Related Sample Code
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201

QTCoreVideo202
Sketch+Accessibility

Declared in
`objc/NSObject.h`

copyScriptingValue:forKey:withProperties:

Creates and returns one or more scripting objects to be inserted into the specified relationship by copying the passed-in value and setting the properties in the copied object or objects.

```
- (id)copyScriptingValue:(id)value forKey:(NSString *)key withProperties:(NSDictionary
 *)properties
```

Parameters

`value`

An object or objects to be copied. The type must match the type of the property identified by `key`. (See also the Discussion section.)

For example, if the property is a to-many relationship, `value` will always be an array of objects to be copied, and this method must therefore return an array of objects.

`key`

A key that identifies the relationship into which to insert the copied object or objects.

`properties`

The properties to be set in the copied object or objects. Derived from the "with properties" parameter of a `duplicate` command. (See also the Discussion section.)

Return Value

The copied object or objects. Returns `nil` if an error occurs.

Discussion

You can override the `copyScriptingValue` method to take more control when your application is sent a `duplicate` command. This method is invoked on the prospective container of the copied object or objects. The `properties` are derived from the `with properties` parameter of the `duplicate` command. The returned objects or objects are then inserted into the container using key-value coding.

When this method is invoked by Cocoa, neither the value nor the properties will have yet been coerced using the `NSScriptKeyValueCoding` method `coerceValue:forKey:`. For sdef-declared scriptability, however, the types of the passed-in objects reliably match the relevant sdef declarations.

The default implementation of this method copies scripting objects by sending `copyWithZone:` to the object or objects specified by `value`. You override this method for situations where this is not sufficient, such as in Core Data applications, in which new objects must be initialized with `[NSManagedObject initWithEntity:insertIntoManagedObjectContext:]`.

Availability
Available in OS X v10.5 and later.

Declared in
`NSObjectScripting.h`

dealloc

Deallocates the memory occupied by the receiver.

```
- (void)dealloc
```

Discussion
Subsequent messages to the receiver may generate an error indicating that a message was sent to a deallocated object (provided the deallocated memory hasn't been reused yet).

You override this method to dispose of resources other than the object's instance variables, for example:

```
- (void)dealloc {
    free(myBigBlockOfMemory);
}
```

In an implementation of `dealloc`, do not invoke the superclass's implementation. You should try to avoid managing the lifetime of limited resources such as file descriptors using `dealloc`.

You never send a `dealloc` message directly. Instead, an object's `dealloc` method is invoked by the runtime. See *Advanced Memory Management Programming Guide* for more details.

Special Considerations
When not using ARC, your implementation of `dealloc` must invoke the superclass's implementation as its last instruction.

Availability
Available in OS X v10.0 and later.

Related Sample Code
From A View to A Movie
From A View to A Picture

QTCoreVideo102
QTCoreVideo201
QTCoreVideo202

Declared in
`objc/NSObject.h`

doesNotRecognizeSelector:

Handles messages the receiver doesn't recognize.

```
- (void)doesNotRecognizeSelector:(SEL)aSelector
```

Parameters
`aSelector`
A selector that identifies a method not implemented or recognized by the receiver.

Discussion
The runtime system invokes this method whenever an object receives an `aSelector` message it can't respond to or forward. This method, in turn, raises an `NSInvalidArgumentException`, and generates an error message.

Any `doesNotRecognizeSelector:` messages are generally sent only by the runtime system. However, they can be used in program code to prevent a method from being inherited. For example, an `NSObject` subclass might renounce the `copy` (page 35) or `init` (page 43) method by re-implementing it to include a `doesNotRecognizeSelector:` message as follows:

```
- (id)copy
{
    [self doesNotRecognizeSelector:_cmd];
}
```

The `_cmd` variable is a hidden argument passed to every method that is the current selector; in this example, it identifies the selector for the `copy` method. This code prevents instances of the subclass from responding to `copy` messages or superclasses from forwarding `copy` messages—although `respondsToSelector:` will still report that the receiver has access to a `copy` method.

If you override this method, you must call `super` or raise an `NSInvalidArgumentException` exception at the end of your implementation. In other words, this method must not return normally; it must always result in an exception being thrown.

Availability
Available in OS X v10.0 and later.

See Also
– `forwardInvocation:` (page 41)

Related Sample Code
AVCustomEditOSX

AVSimpleEditorOSX

Declared in
`objc/NSObject.h`

finalize

The garbage collector invokes this method on the receiver before disposing of the memory it uses. (Deprecated. Garbage collection is deprecated in OS X v10.8; instead, you should use Automatic Reference Counting–see Transitioning to ARC Release Notes .)

```
– (void)finalize
```

Discussion
The garbage collector invokes this method on the receiver before disposing of the memory it uses. When garbage collection is enabled, this method is invoked instead of `dealloc`.

You can override this method to relinquish resources the receiver has obtained, as shown in the following example:

```
– (void)finalize {
    if (log_file != NULL) {
        fclose(log_file);
        log_file = NULL;
    }
    [super finalize];
}
```

Typically, however, you are encouraged to relinquish resources prior to finalization if at all possible. For more details, see “Implementing a finalize Method”.

Special Considerations

It is an error to store `self` into a new or existing live object (colloquially known as "resurrection"), which implies that this method will be called only once. However, the receiver may be messaged after finalization by other objects also being finalized at this time, so your override should guard against future use of resources that have been reclaimed, as shown by the `log_file = NULL` statement in the example. The `finalize` method itself will never be invoked more than once for a given object.

> **Important:** `finalize` methods must be thread-safe.

Availability

Available in OS X v10.4 and later.

See Also

– `dealloc` (page 37)

Related Sample Code

TextInputView

ZipBrowser

Declared in

`objc/NSObject.h`

forwardingTargetForSelector:

Returns the object to which unrecognized messages should first be directed.

```
- (id)forwardingTargetForSelector:(SEL)aSelector
```

Parameters

`aSelector`

A selector for a method that the receiver does not implement.

Return Value

The object to which unrecognized messages should first be directed.

Discussion

If an object implements (or inherits) this method, and returns a non-`nil` (and non-`self`) result, that returned object is used as the new receiver object and the message dispatch resumes to that new object. (Obviously if you return `self` from this method, the code would just fall into an infinite loop.)

If you implement this method in a non-root class, if your class has nothing to return for the given selector then you should return the result of invoking super's implementation.

This method gives an object a chance to redirect an unknown message sent to it before the much more expensive `forwardInvocation:` (page 41) machinery takes over. This is useful when you simply want to redirect messages to another object and can be an order of magnitude faster than regular forwarding. It is not useful where the goal of the forwarding is to capture the NSInvocation, or manipulate the arguments or return value during the forwarding.

Availability
Available in OS X v10.5 and later.

Declared in
`objc/NSObject.h`

forwardInvocation:

Overridden by subclasses to forward messages to other objects.

```
- (void)forwardInvocation:(NSInvocation *)anInvocation
```

Parameters
`anInvocation`
The invocation to forward.

Discussion
When an object is sent a message for which it has no corresponding method, the runtime system gives the receiver an opportunity to delegate the message to another receiver. It delegates the message by creating an `NSInvocation` object representing the message and sending the receiver a `forwardInvocation:` message containing this `NSInvocation` object as the argument. The receiver's `forwardInvocation:` method can then choose to forward the message to another object. (If that object can't respond to the message either, it too will be given a chance to forward it.)

The `forwardInvocation:` message thus allows an object to establish relationships with other objects that will, for certain messages, act on its behalf. The forwarding object is, in a sense, able to "inherit" some of the characteristics of the object it forwards the message to.

> **Important:** To respond to methods that your object does not itself recognize, you must override `methodSignatureForSelector:` (page 45) in addition to `forwardInvocation:`. The mechanism for forwarding messages uses information obtained from `methodSignatureForSelector:` (page 45) to create the `NSInvocation` object to be forwarded. Your overriding method must provide an appropriate method signature for the given selector, either by pre formulating one or by asking another object for one.

An implementation of the `forwardInvocation:` method has two tasks:

- To locate an object that can respond to the message encoded in `anInvocation`. This object need not be the same for all messages.
- To send the message to that object using `anInvocation`. `anInvocation` will hold the result, and the runtime system will extract and deliver this result to the original sender.

In the simple case, in which an object forwards messages to just one destination (such as the hypothetical `friend` instance variable in the example below), a `forwardInvocation:` method could be as simple as this:

```
- (void)forwardInvocation:(NSInvocation *)invocation
{
    SEL aSelector = [invocation selector];

    if ([friend respondsToSelector:aSelector])
        [invocation invokeWithTarget:friend];
    else
        [super forwardInvocation:invocation];
}
```

The message that's forwarded must have a fixed number of arguments; variable numbers of arguments (in the style of `printf()`) are not supported.

The return value of the forwarded message is returned to the original sender. All types of return values can be delivered to the sender: `id` types, structures, double-precision floating-point numbers.

Implementations of the `forwardInvocation:` method can do more than just forward messages. `forwardInvocation:` can, for example, be used to consolidate code that responds to a variety of different messages, thus avoiding the necessity of having to write a separate method for each selector. A `forwardInvocation:` method might also involve several other objects in the response to a given message, rather than forward it to just one.

`NSObject`'s implementation of `forwardInvocation:` simply invokes the `doesNotRecognizeSelector:` (page 38) method; it doesn't forward any messages. Thus, if you choose not to implement `forwardInvocation:`, sending unrecognized messages to objects will raise exceptions.

Availability
Available in OS X v10.0 and later.

Declared in
`objc/NSObject.h`

init

Implemented by subclasses to initialize a new object (the receiver) immediately after memory for it has been allocated.

```
- (id)init
```

Return Value
An initialized object, or `nil` if an object could not be created for some reason that would not result in an exception.

Discussion
An `init` message is coupled with an `alloc` (page 11) (or `allocWithZone:` (page 12)) message in the same line of code:

```
TheClass *newObject = [[TheClass alloc] init];
```

An object isn't ready to be used until it has been initialized. The `init` method defined in the `NSObject` class does no initialization; it simply returns `self`.

In a custom implementation of this method, you must invoke super's designated initializer then initialize and return the new object. If the new object can't be initialized, the method should return `nil`. For example, a hypothetical `BuiltInCamera` class might return `nil` from its `init` method if run on a device that has no camera.

```
- (id)init {
    self = [super init];
    if (self) {
        // Initialize self.
    }
    return self;
}
```

In some cases, an `init` method might return a substitute object. You must therefore always use the object returned by `init`, and not the one returned by `alloc` (page 11) or `allocWithZone:` (page 12), in subsequent code.

Availability
Available in OS X v10.0 and later.

Related Sample Code
From A View to A Movie
QTCoreVideo102
QTCoreVideo103
QTCoreVideo201
QTCoreVideo202

Declared in
`objc/NSObject.h`

inverseForRelationshipKey:

For a given key that defines the name of the relationship from the receiver's class to another class, returns the name of the relationship from the other class to the receiver's class.

```
- (NSString *)inverseForRelationshipKey:(NSString *)relationshipKey
```

Parameters
`relationshipKey`
The name of the relationship from the receiver's class to another class.

Return Value
The name of the relationship that is the inverse of the receiver's relationship named `relationshipKey`.

Discussion
`NSObject`'s implementation of `inverseForRelationshipKey:` simply invokes `[[self classDescription] inverseForRelationshipKey:relationshipKey]`. To make use of the default implementation, you must therefore implement and register a suitable class description—see `NSClassDescription`.

For example, suppose an Employee class has a relationship named `department` to a Department class, and that Department has a relationship called `employees` to Employee. The statement:

```
employee inverseForRelationshipKey:@"department"];
```

returns the string `employees`.

Availability
Available in OS X v10.0 and later.

See Also
– `attributeKeys` (page 29)

– classDescription (page 32)
– toManyRelationshipKeys (page 62)
– toOneRelationshipKeys (page 63)

Declared in
NSClassDescription.h

methodForSelector:

Locates and returns the address of the receiver's implementation of a method so it can be called as a function.

```
– (IMP)methodForSelector:(SEL)aSelector
```

Parameters
aSelector

A selector that identifies the method for which to return the implementation address. The selector must be a valid and non-NULL. If in doubt, use the respondsToSelector: method to check before passing the selector to methodForSelector:.

Return Value
The address of the receiver's implementation of the aSelector.

Discussion
If the receiver is an instance, aSelector should refer to an instance method; if the receiver is a class, it should refer to a class method.

Availability
Available in OS X v10.0 and later.

See Also
+ instanceMethodForSelector: (page 20)

Declared in
objc/NSObject.h

methodSignatureForSelector:

Returns an NSMethodSignature object that contains a description of the method identified by a given selector.

```
– (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
```

Parameters

`aSelector`

A selector that identifies the method for which to return the implementation address. When the receiver is an instance, `aSelector` should identify an instance method; when the receiver is a class, it should identify a class method.

Return Value

An `NSMethodSignature` object that contains a description of the method identified by `aSelector`, or `nil` if the method can't be found.

Discussion

This method is used in the implementation of protocols. This method is also used in situations where an `NSInvocation` object must be created, such as during message forwarding. If your object maintains a delegate or is capable of handling messages that it does not directly implement, you should override this method to return an appropriate method signature.

Availability

Available in OS X v10.0 and later.

See Also

`+ instanceMethodSignatureForSelector:` (page 21)
`– forwardInvocation:` (page 41)

Declared in

`objc/NSObject.h`

mutableCopy

Returns the object returned by `mutableCopyWithZone:` where the zone is `nil`.

```
– (id)mutableCopy
```

Return Value

The object returned by the `NSMutableCopying` protocol method `mutableCopyWithZone:`, where the zone is `nil`.

Discussion

This is a convenience method for classes that adopt the `NSMutableCopying` protocol. An exception is raised if there is no implementation for `mutableCopyWithZone:`.

Availability

Available in OS X v10.0 and later.

Related Sample Code
DesktopImage
IdentitySample
iSpend
Sketch+Accessibility
TextEdit

Declared in
`objc/NSObject.h`

newScriptingObjectOfClass:forValueForKey:withContentsValue:properties:

Creates and returns an instance of a scriptable class, setting its contents and properties, for insertion into the relationship identified by the key.

```
- (id)newScriptingObjectOfClass:(Class)class forValueForKey:(NSString *)key
withContentsValue:(id)contentsValue properties:(NSDictionary *)properties
```

Parameters

`class`
The class of the scriptable object to be created.

`key`
A key that identifies the relationship into which the new class object will be inserted.

`contentsValue`
Specifies the contents of the object to be created. This may be `nil`. (See also the Discussion section.)

`properties`
The properties to be set in the new object. (See also the Discussion section.)

Return Value
The new object. Returns `nil` if an error occurs.

Discussion
You can override the `newScriptingObjectOfClass` method to take more control when your application is sent a `make` command. This method is invoked on the prospective container of the new object.
The `contentsValue` and `properties` are derived from the `with contents` and `with properties` parameters of the `make` command. The returned objects or objects are then inserted into the container using key-value coding.

When this method is invoked by Cocoa, neither the contents value nor the properties will have yet been coerced using the `NSScriptKeyValueCoding` method `coerceValue:forKey:`. For sdef-declared scriptability, however, the types of the passed-in objects reliably match the relevant sdef declarations.

The default implementation of this method creates new scripting objects by sending `alloc` to a class and `init` to the resulting object. You override this method for situations where this is not sufficient, such as in Core Data applications, in which new objects must be initialized with `[NSManagedObject initWithEntity:insertIntoManagedObjectContext:]`.

Availability
Available in OS X v10.5 and later.

Declared in
`NSObjectScripting.h`

performSelector:onThread:withObject:waitUntilDone:

Invokes a method of the receiver on the specified thread using the default mode.

```
- (void)performSelector:(SEL)aSelector onThread:(NSThread *)thread withObject:(id)arg
 waitUntilDone:(BOOL)wait
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`thread`

The thread on which to execute `aSelector`.

`arg`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`wait`

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the specified thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread and target thread are the same, and you specify `YES` for this parameter, the selector is performed immediately on the current thread. If you specify `NO`, this method queues the message on the thread's run loop and returns, just like it does for other threads. The current thread must then dequeue and process the message when it has an opportunity to do so.

Discussion
You can use this method to deliver messages to other threads in your application. The message in this case is a method of the current object that you want to execute on the target thread.

This method queues the message on the run loop of the target thread using the default run loop modes—that is, the modes associated with the `NSRunLoopCommonModes` constant. As part of its normal run loop processing, the target thread dequeues the message (assuming it is running in one of the default run loop modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 51) or `performSelector:withObject:afterDelay:inModes:` (page 52) method.

Special Considerations
This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability
Available in OS X v10.5 and later.

See Also
– `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)
– `performSelectorInBackground:withObject:` (page 53)

Declared in
`NSThread.h`

performSelector:onThread:withObject:waitUntilDone:modes:

Invokes a method of the receiver on the specified thread using the specified modes.

```
- (void)performSelector:(SEL)aSelector onThread:(NSThread *)thread withObject:(id)arg
 waitUntilDone:(BOOL)wait modes:(NSArray *)array
```

Parameters
`aSelector`
A selector that identifies the method to invoke. It should not have a significant return value and should take a single argument of type `id`, or no arguments.

`thread`
The thread on which to execute `aSelector`. This thread represents the target thread.

`arg`
The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`wait`

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the specified thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread and target thread are the same, and you specify `YES` for this parameter, the selector is performed immediately. If you specify `NO`, this method queues the message and returns immediately, regardless of whether the threads are the same or different.

`array`

An array of strings that identifies the modes in which it is permissible to perform the specified selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector. For information about run loop modes, see "Run Loops" in *Threading Programming Guide*.

Discussion

You can use this method to deliver messages to other threads in your application. The message in this case is a method of the current object that you want to execute on the target thread.

This method queues the message on the run loop of the target thread using the run loop modes specified in the `array` parameter. As part of its normal run loop processing, the target thread dequeues the message (assuming it is running in one of the specified modes) and invokes the desired method.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 51) or `performSelector:withObject:afterDelay:inModes:` (page 52) method instead.

Special Considerations

This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability

Available in OS X v10.5 and later.

See Also

– `performSelector:onThread:withObject:waitUntilDone:` (page 48)
– `performSelectorInBackground:withObject:` (page 53)

Declared in

`NSThread.h`

performSelector:withObject:afterDelay:

Invokes a method of the receiver on the current thread using the default mode after a delay.

```
- (void)performSelector:(SEL)aSelector withObject:(id)anArgument
afterDelay:(NSTimeInterval)delay
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`anArgument`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`delay`

The minimum time before which the message is sent. Specifying a delay of 0 does not necessarily cause the selector to be performed immediately. The selector is still queued on the thread's run loop and performed as soon as possible.

Discussion

This method sets up a timer to perform the `aSelector` message on the current thread's run loop. The timer is configured to run in the default mode (`NSDefaultRunLoopMode`). When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in the default mode; otherwise, the timer waits until the run loop is in the default mode.

If you want the message to be dequeued when the run loop is in a mode other than the default mode, use the `performSelector:withObject:afterDelay:inModes:` (page 52) method instead. If you are not sure whether the current thread is the main thread, you can use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56) method to guarantee that your selector executes on the main thread. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 13) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 14) method.

Special Considerations

This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability

Available in OS X v10.0 and later.

See Also

+ `cancelPreviousPerformRequestsWithTarget:selector:object:` (page 14)
– `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54)
– `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56)
– `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

Related Sample Code
AttachAScript
AVReaderWriter for OSX
iSpend
Quartz Composer RepositoryBrowser
TargetGallery

Declared in
`NSRunLoop.h`

performSelector:withObject:afterDelay:inModes:

Invokes a method of the receiver on the current thread using the specified modes after a delay.

```
– (void)performSelector:(SEL)aSelector withObject:(id)anArgument
afterDelay:(NSTimeInterval)delay inModes:(NSArray *)modes
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`anArgument`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`delay`

The minimum time before which the message is sent. Specifying a delay of 0 does not necessarily cause the selector to be performed immediately. The selector is still queued on the thread's run loop and performed as soon as possible.

`modes`

An array of strings that identify the modes to associate with the timer that performs the selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector. For information about run loop modes, see "Run Loops" in *Threading Programming Guide* .

Discussion

This method sets up a timer to perform the `aSelector` message on the current thread's run loop. The timer is configured to run in the modes specified by the `modes` parameter. When the timer fires, the thread attempts to dequeue the message from the run loop and perform the selector. It succeeds if the run loop is running and in one of the specified modes; otherwise, the timer waits until the run loop is in one of those modes.

If you want the message to be dequeued when the run loop is in a mode other than the default mode, use the `performSelector:withObject:afterDelay:inModes:` (page 52) method instead. If you are not sure whether the current thread is the main thread, you can use the `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54) or `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56) method to guarantee that your selector executes on the main thread. To cancel a queued message, use the cancelPreviousPerformRequestsWithTarget: (page 13) or cancelPreviousPerformRequestsWithTarget:selector:object: (page 14) method.

Special Considerations

This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability

Available in OS X v10.0 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 51)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

`addTimer:forMode:` (`NSRunLoop`)
`invalidate` (`NSTimer`)

Declared in

`NSRunLoop.h`

performSelectorInBackground:withObject:

Invokes a method of the receiver on a new background thread.

```
- (void)performSelectorInBackground:(SEL)aSelector withObject:(id)arg
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`arg`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

Discussion

This method creates a new thread in your application, putting your application into multithreaded mode if it was not already. The method represented by `aSelector` must set up the thread environment just as you would for any other new thread in your program. For more information about how to configure and run threads, see *Threading Programming Guide* .

Availability

Available in OS X v10.5 and later.

See Also

`– performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

Declared in

`NSThread.h`

performSelectorOnMainThread:withObject:waitUntilDone:

Invokes a method of the receiver on the main thread using the default mode.

```
– (void)performSelectorOnMainThread:(SEL)aSelector withObject:(id)arg
waitUntilDone:(BOOL)wait
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`arg`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`wait`

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the main thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread is also the main thread, and you specify `YES` for this parameter, the message is delivered and processed immediately.

Discussion

You can use this method to deliver messages to the main thread of your application. The main thread encompasses the application's main run loop, and is where the `NSApplication` object receives events. The message in this case is a method of the current object that you want to execute on the thread.

This method queues the message on the run loop of the main thread using the common run loop modes—that is, the modes associated with the `NSRunLoopCommonModes` constant. As part of its normal run loop processing, the main thread dequeues the message (assuming it is running in one of the common run loop modes) and invokes the desired method. Multiple calls to this method from the same thread cause the corresponding selectors to be queued and performed in the same same order in which the calls were made.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 51) or `performSelector:withObject:afterDelay:inModes:` (page 52) method.

Special Considerations

This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability

Available in OS X v10.2 and later.

See Also

- `performSelector:withObject:afterDelay:` (page 51)
- `performSelector:withObject:afterDelay:inModes:` (page 52)
- `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

Related Sample Code

CocoaDVDPlayer

CocoaSpeechSynthesisExample

TextEdit

VirtualScanner

ZipBrowser

Declared in
NSThread.h

performSelectorOnMainThread:withObject:waitUntilDone:modes:

Invokes a method of the receiver on the main thread using the specified modes.

```
- (void)performSelectorOnMainThread:(SEL)aSelector withObject:(id)arg
waitUntilDone:(BOOL)wait modes:(NSArray *)array
```

Parameters

`aSelector`

A selector that identifies the method to invoke. The method should not have a significant return value and should take a single argument of type `id`, or no arguments.

`arg`

The argument to pass to the method when it is invoked. Pass `nil` if the method does not take an argument.

`wait`

A Boolean that specifies whether the current thread blocks until after the specified selector is performed on the receiver on the main thread. Specify `YES` to block this thread; otherwise, specify `NO` to have this method return immediately.

If the current thread is also the main thread, and you pass `YES`, the message is performed immediately, otherwise the perform is queued to run the next time through the run loop.

`array`

An array of strings that identifies the modes in which it is permissible to perform the specified selector. This array must contain at least one string. If you specify `nil` or an empty array for this parameter, this method returns without performing the specified selector. For information about run loop modes, see "Run Loops" in *Threading Programming Guide*.

Discussion

You can use this method to deliver messages to the main thread of your application. The main thread encompasses the application's main run loop, and is where the `NSApplication` object receives events. The message in this case is a method of the current object that you want to execute on the thread.

This method queues the message on the run loop of the main thread using the run loop modes specified in the `array` parameter. As part of its normal run loop processing, the main thread dequeues the message (assuming it is running in one of the specified modes) and invokes the desired method. Multiple calls to this method from the same thread cause the corresponding selectors to be queued and performed in the same

same order in which the calls were made, assuming the associated run loop modes for each selector are the same. If you specify different modes for each selector, any selectors whose associated mode does not match the current run loop mode are skipped until the run loop subsequently executes in that mode.

You cannot cancel messages queued using this method. If you want the option of canceling a message on the current thread, you must use either the `performSelector:withObject:afterDelay:` (page 51) or `performSelector:withObject:afterDelay:inModes:` (page 52) method.

Special Considerations
This method registers with the runloop of its current context, and depends on that runloop being run on a regular basis to perform correctly. One common context where you might call this method and end up registering with a runloop that is not automatically run on a regular basis is when being invoked by a dispatch queue. If you need this type of functionality when running on a dispatch queue, you should use `dispatch_after` and related methods to get the behavior you want.

Availability
Available in OS X v10.2 and later.

See Also
- `performSelector:withObject:afterDelay:` (page 51)
- `performSelector:withObject:afterDelay:inModes:` (page 52)
- `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54)
- `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49)

Declared in
`NSThread.h`

replacementObjectForArchiver:

Overridden by subclasses to substitute another object for itself during archiving.

```
- (id)replacementObjectForArchiver:(NSArchiver *)anArchiver
```

Parameters
`anArchiver`
The archiver creating an archive.

Return Value
The object to substitute for the receiver during archiving.

Discussion
This method is invoked by `NSArchiver`. `NSObject`'s implementation returns the object returned by `replacementObjectForCoder:` (page 58).

Availability
Available in OS X v10.0 and later.

See Also
– `classForArchiver` (page 32)

Declared in
`NSArchiver.h`

replacementObjectForCoder:

Overridden by subclasses to substitute another object for itself during encoding.

```
- (id)replacementObjectForCoder:(NSCoder *)aCoder
```

Parameters
`aCoder`

The coder encoding the receiver.

Return Value
The object encode instead of the receiver (if different).

Discussion
An object might encode itself into an archive, but encode a proxy for itself if it's being encoded for distribution. This method is invoked by `NSCoder`. `NSObject`'s implementation returns `self`.

Availability
Available in OS X v10.0 and later.

See Also
– `classForCoder` (page 33)
– `awakeAfterUsingCoder:` (page 30)

Declared in
`NSObject.h`

replacementObjectForKeyedArchiver:

Overridden by subclasses to substitute another object for itself during keyed archiving.

```
- (id)replacementObjectForKeyedArchiver:(NSKeyedArchiver *)archiver
```

Parameters

`archiver`

A keyed archiver creating an archive.

Return Value

The object encode instead of the receiver (if different).

Discussion

This method is called only if no replacement mapping for the object has been set up in the encoder (for example, due to a previous call of `replacementObjectForKeyedArchiver:` to that object).

Availability

Available in OS X v10.2 and later.

See Also

– `classForKeyedArchiver` (page 33)

Declared in

`NSKeyedArchiver.h`

replacementObjectForPortCoder:

Overridden by subclasses to substitute another object or a copy for itself during distribution encoding.

```
- (id)replacementObjectForPortCoder:(NSPortCoder *)aCoder
```

Parameters

`aCoder`

The port coder encoding the receiver.

Return Value

The object encode instead of the receiver (if different).

Discussion

This method is invoked by `NSPortCoder`. `NSObject`'s implementation returns an `NSDistantObject` object for the object returned by `replacementObjectForCoder:` (page 58), enabling all objects to be distributed by proxy as the default. However, if `replacementObjectForCoder:` (page 58) returns `nil`, `NSObject`'s implementation will also return `nil`.

Subclasses that want to be passed by copy instead of by reference must override this method and return `self`. The following example shows how to support object replacement both by copy and by reference:

```
- (id)replacementObjectForPortCoder:(NSPortCoder *)encoder {
```

```
    return [encoder isByref] ? [super replacementObjectForPortCoder:encoder] :
self;
}
```

Availability
Available in OS X v10.0 and later.

See Also
– `classForPortCoder` (page 34)

Declared in
`NSPortCoder.h`

scriptingProperties

Returns an `NSString`-keyed dictionary of the receiver's scriptable properties.

```
– (NSDictionary *)scriptingProperties
```

Return Value
An `NSString`-keyed dictionary of the receiver's scriptable properties, including all of those that are declared as Attributes and ToOneRelationships in the `.scriptSuite` property list entries for the class and its scripting superclasses, with the exception of ones keyed by "scriptingProperties." Each key in the dictionary must be identical to the key for an Attribute or ToOneRelationship. The values of the dictionary must be Objective-C objects that are convertible to `NSAppleEventDescriptor` objects.

Availability
Available in OS X v10.2 and later.

See Also
– `setScriptingProperties:` (page 61)

Declared in
`NSObjectScripting.h`

scriptingValueForSpecifier:

Given an object specifier, returns the specified object or objects in the receiving container.

```
– (id)scriptingValueForSpecifier:(NSScriptObjectSpecifier *)objectSpecifier
```

Parameters

`objectSpecifier`

An object specifier to be evaluated.

Return Value

The specified object or objects in the receiving container.

This method might successfully return an object, an array of objects, or `nil`, depending on the kind of object specifier. Because `nil` is a valid return value, failure is signaled by invoking the object specifier's `setEvaluationError:` method before returning.

Discussion

You can override this method to customize the evaluation of object specifiers without requiring that the scripting container make up indexes for contained objects that don't naturally have indexes (as can be the case if you implement `indicesOfObjectsByEvaluatingObjectSpecifier:` instead).

Your override of this method doesn't need to also invoke any of the `NSScriptCommand` error signaling methods, though it can, to record very specific information. The `NSUnknownKeySpecifierError` and `NSInvalidIndexSpecifierError` numbers are special, in that Cocoa may continue evaluating an outer specifier if they're encountered, for the convenience of scripters.

Availability

Available in OS X v10.5 and later.

Declared in

`NSObjectScripting.h`

setScriptingProperties:

Given an `NSString`-keyed dictionary, sets one or more scriptable properties of the receiver.

```
- (void)setScriptingProperties:(NSDictionary *)properties
```

Parameters

`properties`

A dictionary containing one or more scriptable properties of the receiver. The valid keys for the dictionary include the keys for non-ReadOnly Attributes and ToOneRelationships in the `.scriptSuite` property list entries for the object's class and its scripting superclasses, and no others. The values of the dictionary are Objective-C objects.

Discussion

Invokers of this method must have already done any necessary validation to ensure that the properties dictionary includes nothing but entries for declared, settable, Attributes and ToOneRelationships. Implementations of this method are not expected to check the validity of keys in the passed-in dictionary, but must be able to accept dictionaries that do not contain entries for every scriptable property. Implementations of this method must perform type checking on the dictionary values.

Availability

Available in OS X v10.2 and later.

See Also

– `scriptingProperties` (page 60)

Declared in

`NSObjectScripting.h`

toManyRelationshipKeys

Returns array containing the keys for the to-many relationship properties of the receiver.

```
– (NSArray *)toManyRelationshipKeys
```

Return Value

An array containing the keys for the to-many relationship properties of the receiver (if any).

Discussion

`NSObject`'s implementation simply invokes `[[self classDescription] toManyRelationshipKeys]`. To make use of the default implementation, you must therefore implement and register a suitable class description—see `NSClassDescription`.

Availability

Available in OS X v10.0 and later.

See Also

– `attributeKeys` (page 29)
– `classDescription` (page 32)
– `inverseForRelationshipKey:` (page 44)
– `toOneRelationshipKeys` (page 63)

Declared in

`NSClassDescription.h`

toOneRelationshipKeys

Returns the keys for the to-one relationship properties of the receiver, if any.

```
- (NSArray *)toOneRelationshipKeys
```

Return Value

An array containing the keys for the to-one relationship properties of the receiver.

Discussion

`NSObject`'s implementation of `toOneRelationshipKeys` simply invokes `[[self classDescription] toOneRelationshipKeys]`. To make use of the default implementation, you must therefore implement and register a suitable class description—see `NSClassDescription`.

Availability

Available in OS X v10.0 and later.

See Also

- `attributeKeys` (page 29)
- `classDescription` (page 32)
- `toManyRelationshipKeys` (page 62)
- `inverseForRelationshipKey:` (page 44)

Declared in

`NSClassDescription.h`

Deprecated NSObject Methods

A method identified as deprecated has been superseded and may become unsupported in the future.

Deprecated in OS X v10.5

poseAsClass:

Causes the receiving class to pose as a specified superclass. (Deprecated in OS X v10.5.)

```
+ (void)poseAsClass:(Class)aClass
```

Parameters

`aClass`

A superclass of the receiver.

Discussion

The receiver takes the place of `aClass` in the inheritance hierarchy; all messages sent to `aClass` will actually be delivered to the receiver. The receiver must be defined as a subclass of `aClass`. It can't declare any new instance variables of its own, but it can define new methods and override methods defined in `aClass`. The `poseAsClass:` message should be sent before any messages are sent to `aClass` and before any instances of `aClass` are created.

This facility allows you to add methods to an existing class by defining them in a subclass and having the subclass substitute for the existing class. The new method definitions will be inherited by all subclasses of the superclass. Care should be taken to ensure that the inherited methods do not generate errors.

A subclass that poses as its superclass still inherits from the superclass. Therefore, none of the functionality of the superclass is lost in the substitution. Posing doesn't alter the definition of either class.

Posing is useful as a debugging tool, but category definitions are a less complicated and more efficient way of augmenting existing classes. Posing admits only two possibilities that are absent from categories:

- A method defined by a posing class can override any method defined by its superclass. Methods defined in categories can replace methods defined in the class proper, but they cannot reliably replace methods defined in other categories. If two categories define the same method, one of the definitions will prevail, but there's no guarantee which one.

- A method defined by a posing class can, through a message to `super`, incorporate the superclass method it overrides. A method defined in a category can replace a method defined elsewhere by the class, but it can't incorporate the method it replaces.

Special Considerations

Posing is deprecated in OS X v10.5. The `poseAsClass:` method is not available in 64-bit applications in OS X v10.5.

Availability

Available in OS X v10.0 and later.

Deprecated in OS X v10.5.

Not available to 64-bit applications.

Declared in

`NSObject.h`

Document Revision History

This table describes the changes to *NSObject Class Reference*.

Date	Notes
2013-12-16	Made minor clarifications concerning initialization and performing selectors. Corrected description of `initialize` (page 19) method to describe actual behavior and remove spurious recommendation to not call `super`. Added recommendation to avoid complex initialization code, especially code that might take locks. Added note to `performSelector:withObject:afterDelay:` (page 51), `performSelector:withObject:afterDelay:inModes:` (page 52), `performSelectorOnMainThread:withObject:waitUntilDone:` (page 54), `performSelectorOnMainThread:withObject:waitUntilDone:modes:` (page 56), `performSelector:onThread:withObject:waitUntilDone:` (page 48), `performSelector:onThread:withObject:waitUntilDone:modes:` (page 49), and `performSelectorInBackground:withObject:` (page 53) indicating that they may not run in a reasonable timeframe if called in the context of a dispatch queue, or any other context where a runloop exists but may not be run on a regular basis. Added an example to `init` (page 43) of a case where an implementation might reasonably return `nil`.
2013-10-22	Updated code snippet in replacementObjectForPortCoder: to use current API.
2013-09-17	Added links to functions related to class objects.
2013-01-28	Updated description of forwardInvocation: to recommend calling super instead of doesNotRecognizeSelector:.

Date	Notes
2012-09-19	Updated for OS X v10.8.
2012-02-16	Updated discussion of alloc methods for the modern runtime.
2011-09-08	Updated for ARC.
2011-01-19	Updated the performSelectorOnMainThread... method descriptions to reflect what happens during sequential calls to the method from the same thread.
2010-03-24	Update example code to new initializer pattern.
2009-08-28	Added autoContentAccessingProxy method.
2009-03-24	Updated for OS X v10.6.
2009-02-04	Added note about managing scarce resources in dealloc.
2008-10-15	Added special consideration for +initialize method.
2008-06-09	Updated sample code in +resolveInstanceMethod: to properly call super.
2008-03-11	Updated the descriptions of the methods copyScriptingValue:forKey:withProperties: and newScriptingObjectOfClass:forValueForKey: withContentsValue:properties:.
2008-02-08	Updated the description of the load method.
2007-12-11	Updated descriptions for the performSelector methods.
2007-07-07	Included new API for OS X v10.5. The following new methods have been added for use with Cocoa scripting: `copyScriptingValue:forKey:withProperties:` (page 36), `newScriptingObjectOfClass:forValueForKey:withContentsValue:properties:` (page 47), and `scriptingValueForSpecifier:` (page 60).
2006-06-28	Augmented the description of dealloc and version.

Date	Notes
2006-05-23	First publication of this content as a separate document.

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